Escuela Politécnica Superior

Videogames for a living

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
José Manuel Palau Alegría

Tutora:
María Violeta Migallón Gomis

Diciembre 2018

Universitat d'Alacant
Universidad de Alicante

“Todos somos muy ignorantes, lo que ocurre es que no todos ignoramos las mismas cosas“.

Albert Einstein.

“The wind is howling. You're filled with determination…”

Toby Fox, Undertale.

Agradecimientos

Este trabajo, y el que haya realizado este grado no habría sido posible sin el apoyo de mis padres, que han estado ahí desde siempre, apoyándome y dándome ánimos, y como dice mi madre, "confío en ti más que tú mismo".

Gracias también a mi pareja, por aguantarme, quererme, pincharme para que me pusiese a trabajar en este proyecto, y apoyarme en esta travesía.

A mis amigas y amigos, por estar ahí apoyándome siempre. ¿Veis como estaba ya casi terminado?

Por supuesto, a mi tutora, Violeta por implicarse tantísimo y ayudarme tanto.

Y por último, pero no por ello menos importante, gracias a mi familia, en su totalidad, gracias por tanto, en especial a mis abuelos. Esto va por vosotras y por vosotros.

ÍNDICE

RESUMEN

Actualmente, las grandes compañías de videojuegos tienen sistemas para analizar diferentes factores que afectan a la diversión de un videojuego, principalmente mediante distintos tipos de métricas. Estas métricas son muy útiles a la hora de conocer cómo interactúan los jugadores con el videojuego y detectar qué aspectos se pueden mejorar.

Este trabajo tiene como objetivo principal acercar la posibilidad de realizar estas métricas a cualquier desarrollo, sea del tamaño que sea. Para ello se debe diseñar y desarrollar un sistema capaz de recibir información desde cualquier videojuego que tenga conexión a Internet, con todo lo que ello conlleva, es decir, un servidor, con su lógica interna y su base de datos, una web desde la que poder interactuar con la misma, y algunos juegos sencillos para poder probar su funcionamiento.

Y en esto ha consistido “Videogames for a living”, en el desarrollo de dos juegos de móvil en Unity y un sistema web que permite mostrar las diferentes métricas que almacenen los videojuegos que se introduzcan en dicho sistema. Además se incluye un sistema de creación de variables para la obtención de nuevas métricas atendiendo a las características que se desean recopilar de un juego particular.

1. Introducción

Los videojuegos son un arte y como cualquier arte, su valor no radica en algo tangible. Dos más dos son cuatro, pero el mismo trazo, por muy magistral que sea, puede ser lo que le falte a una obra para llegar a ser maestra o para destrozar por completo otra. Los videojuegos también sufren de esto, pero a una escala mucho mayor. Una canción puede no funcionar, pero las pérdidas monetarias y temporales debidas a que falle un videojuego, o una parte del mismo, son astronómicas en comparación.

Se puede dar, por ejemplo, el caso de *Diablo 3* [21], un videojuego en el que el personaje a controlar se puede mejorar con diferentes objetos para equipar, en el que su modo más difícil requería equipo que solamente se podía conseguir terminando dicho modo y, por tanto, era prácticamente imposible de obtener. Esto, sumado a otros errores similares de diseño, hizo que el público tuviera una muy mala opinión de dicho título, lo que normalmente se traduce en malas ventas. Esto se resolvió posteriormente mediante los comentarios de los jugadores respecto a este tema. Según el director de diseño de la versión de consola del mismo videojuego, Josh Mosqueira [72], “Puedes desarrollar un juego, puedes probarlo, y puedes pensar que sabes que conoces el juego hasta que lo lanzas, pero el primer día de lanzamiento, lo más probable es que los jugadores hayan dedicado más horas a jugarlo que toda la duración del desarrollo, por lo que vas a ver cosas en las que nunca pensaste”.

Por tanto, las empresas desarrolladoras de videojuegos tienen que tener información sobre qué va a funcionar, o en términos de videojuego, qué va a ser divertido y atractivo, y qué no. Y qué mejor manera que aprovechar las miles de horas que se van a dedicar a dichos videojuegos para recibir esta información de inmediato y en tiempo real. Es por esto, por lo que este trabajo tiene su razón de ser. Las grandes compañías tienen ya sus sistemas para medir diferentes factores que afectan a la diversión de un videojuego, principalmente mediante diferentes tipos de métricas. Sin embargo, los pequeños estudios no tienen normalmente acceso a este tipo de herramientas.

Estas métricas pueden tener una utilidad prácticamente infinita en este terreno como, por ejemplo, ajustar la dificultad a lo que está contando el videojuego, evitando por ejemplo, que el explorar una zona que se le ha dicho al jugador que es altamente peligrosa sea un paseo por el campo para el mismo.

Este proyecto tiene como objetivo principal acercar la posibilidad de realizar estas métricas a cualquier desarrollo, sea del tamaño que sea. Para conseguir este objetivo se debe diseñar y desarrollar un sistema capaz de recibir información desde cualquier videojuego que tenga conexión a Internet, con todo lo que ello conlleva, es decir, un servidor, con su lógica interna y su base de datos, una web desde la que poder interactuar con la misma, y algunos juegos de prueba sencillos para poder probar su funcionamiento. Y esto es lo que se ha realizado en el proyecto "Videogames for a living" y que se explica detalladamente en este documento.

Concretamente, después de esta breve introducción, en el Capítulo 2 se trata el estado del arte del proyecto a desarrollar, tratando el concepto de videojuego, tipos de videojuegos y algunos aspectos relacionados con los mismos, incluyendo las métricas y una breve reseña sobre la actualidad de la industria del videojuego. El Capítulo 3 profundiza en los objetivos tanto generales como específicos que se han pretendido alcanzar con este trabajo. El análisis de requisitos y la especificación del sistema se trata en el Capítulo 4, mientras que en el Capítulo 5 se describen los desarrollos que se han realizado para este proyecto, tanto los videojuegos *Penguin Rush* y *Pizza Clicker,* como el sistema de gestión de métricas desarrollado. Se finaliza con una serie de conclusiones y líneas futuras (véase Capítulo 6), así como con la bibliografía utilizada para la realización del proyecto y elaboración de este documento. Como anexos se incluyen los índices correspondientes a los listados de figuras y tablas.

2. Estado del arte

2.1. Introducción

Actualmente hay discrepancias sobre qué es exactamente un videojuego [23], [62]. ¿Es requerida una interacción del usuario constante para que un sistema se considere videojuego? ¿Debe tener gráficos un videojuego para considerarse como tal? Según la definición que se puede encontrar en Wikipedia [91], un videojuego es “un juego electrónico en el que una o más personas interactúan, por medio de un controlador, con un dispositivo que muestra imágenes de video”, por lo que podemos entender que un videojuego es todo juego electrónico con el que se pueda interactuar y muestre imágenes.

Teniendo en cuenta la definición anterior, a lo largo de este capítulo se expondrá el videojuego y sus partes en detalle, tipos y géneros de videojuegos que se pueden encontrar, haciendo un pequeño inciso en los géneros que se han tocado al realizar este proyecto y algunos aspectos de las métricas en los videojuegos. Además, se hará una breve reseña sobre la actualidad que nos encontramos en la industria del videojuego.

2.2. Tipos y géneros de videojuegos

Los videojuegos pueden ser clasificados mediante tipos y, dentro de los mismos mediante géneros, es decir, grupos de videojuegos que comparten mecánicas, tropos, una estética y jugabilidades similares, como por ejemplo el género plataformas, con una jugabilidad enfocada principalmente al movimiento y los saltos (véase Figura 1), o el tipo de videojuego RPG (Role Play Game), basado en el combate usando estadísticas como ataque, defensa, donde prima la estrategia (véase Figura 2).

Figura 1: *Super Mario Bros, NES.*

Fuente: https://www.nintendo.es/Juegos/NES/Super-Mario-Bros--803853.html.

Figura 2: *Final Fantasy VI Advance, GBA, ejemplo de RPG.*

Fuente: captura del videojuego desde emulador.

No existe una clasificación consensuada y única de videojuegos, debido a la existencia de multitud de factores que pueden tenerse en cuenta a la hora de definir los distintos géneros y las distintas formas de agruparlos [49], [71]. También deberemos considerar que estos géneros pueden tener subgéneros, ya que no es lo mismo un juego de disparos como *Overwatch* [61], basado en partidas rápidas con equipos pequeños, que un *Arma* [5], que entra casi en el terreno de la simulación militar. Estos subgéneros no se tendrán en cuenta en el listado que veremos a continuación. Dicho listado está basado en el artículo que hace una clasificación de los diferentes géneros que encontraríamos en la página de Wikipedia en inglés [46], y teniendo en cuenta agrupaciones de géneros, es decir, tipos de juego.

- **Videojuegos de acción**, que por norma general alientan o requieren de más habilidad y coordinación ojo-mano. En este tipo podríamos agrupar los siguientes géneros:

    - **Plataformas**: basados en llegar de un punto a otro saltando y trepando por los niveles. Un claro ejemplo de este tipo de videojuegos podría ser *Super Mario Bros* [80].
    - **Disparos**: juegos en los que la mecánica principal suele ser disparar, ya sea a enemigos, como en *Half Life* [34] o a las paredes para poder resolver puzles, como en *Portal* [65].
    - **Lucha**: en los que, por norma general dos o más jugadores se enfrentan en un combate igualado. Este género estaría representado de manera muy fiel por *Street Fighter* II [78].
    - **Supervivencia**: juegos en los que el jugador empezará con prácticamente ningún recurso en un mundo hostil y deberá recolectar dichos recursos para sobrevivir, como se puede ver en videojuegos como *Minecraft* [53].
    - **Survival Horror**: juegos que utilizan su medio para transmitir terror al jugador, ya sea con una jugabilidad que permite defenderse, como en *Resident Evil* [68], o en la que la única mecánica sea moverse e interactuar con el entorno, como lo es *Amnesia* [3].

- **Metroidvania**: recibe el nombre de dos franquicias distintas, *Metroid* [52] y *Castlevania* [15], y es un género basado en la premisa de tener un mapa enorme interconectado, pero el acceso a ciertas áreas está bloqueado hasta que se realiza cierta acción o se consigue cierta arma, habilidad o herramienta. Un claro ejemplo de este género de videojuego lo encontramos en *Guacamelee* [33] y *Hollow Knight* [38].
- **Endless runner**: juegos en los que el personaje del jugador se encuentra en un camino infinito y deberá intentar sobrevivir lo máximo posible utilizando diferentes recursos, como pueden ser saltar, moverse a los lados o disparar. El videojuego que se ha realizado para este proyecto, *Penguin Rush* pertenece a este género.

- **Videojuegos de aventura**: su mecánica principal es explorar y resolver puzles. Están muy basados en la narrativa, y suelen no requerir mucha habilidad con los controles, por lo que es un buen punto de entrada para un jugador primerizo. En este tipo podríamos agrupar los siguientes géneros:

  - **Aventuras textuales**: videojuegos arcaicos en base texto, es decir, sin gráficos. La forma de interaccionar con ellos sería escribir y leer, como por ejemplo en *Zork* [93], cuyo inicio es el siguiente:

```
Welcome to ZORK.
Release 13 / Serial number 040826 / Inform v6.14 Library 6/7
West of House
This is an open field west of a white house, with a boarded front door.
There is a small mailbox here.
A rubber mat saying 'Welcome to Zork!' lies by the door.
```

    Y podemos interactuar con el juego diciendo cosas como "*go south*", "*open door*", y el sistema responderá. Por ejemplo, probemos con "*open mailbox"*:

```
>open mailbox
You open the mailbox, revealing a small leaflet.
```

    El sistema detecta que se quiere abrir el correo y da la respuesta apropiada. Esta mecánica, como se puede observar, es un poco limitada.

- **Aventuras gráficas**: juegos de aventura similares a los textuales, es decir, realizan acciones simples como “abrir puerta”, “tirar espada” o “coger garfio”, pero apoyándose en gráficos. El buque insignia de este género sería *Monkey Island* [55] y un ejemplo de aventura gráfica de alta calidad moderna sería *The Wolf Among Us* [84].
- **Novelas visuales**: como su propio nombre indica son juegos basados en leer una historia, pudiendo interactuar con ella de una forma u otra, y utilizando gráficos estáticos. Tendríamos como ejemplo de este género la saga *Ace Attorney* [1].
- **Juegos narrativos**: juegos en los que la historia está muy por encima de las mecánicas, siendo este último una excusa para sumergir al jugador en la historia, como por ejemplo, *Detroit Become Human* [20].
- **Walking simulator**: o simulador de andar, se refiere a los juegos en los que la interacción del usuario se reduce normalmente a pasear e interactuar con el entorno. *The Stanley Parable* [83] es un buen ejemplo de estos.

- **Role Play Games (RPG)**: toman su principal mecánica e inspiración de juegos de rol de mesa como *Dragones y Mazmorras* [22]. Estos juegos, en su mayoría, tienen protagonistas especializados en diferentes habilidades como, por ejemplo, combate cuerpo a cuerpo, utilizar magia o disparar a distancia. Otro de los puntos en común de estos juegos es la progresión del personaje, utilizando normalmente un sistema de experiencia y niveles, mejorando y añadiendo capas de complejidad a los personajes según se avanza en el videojuego. Este tipo de juegos englobaría los siguientes géneros:

  - **RPG de acción**: es un género que bebe tanto de los juegos RPG como de los juegos de acción, tomando mecánicas de ambos. Un claro ejemplo de este género de videojuegos es *Diablo* [21].
  - **MMORPG (Multiplayer Massive Online RPG)**: videojuegos con una base RPG pero que permiten el juego multijugador a miles de personas de manera simultánea. El máximo exponente, aunque no el primero, es *World Of Warcraft* (WoW) [92].

- **Roguelike**: género que toma muchos elementos, e incluso el nombre, del videojuego *Rogue* [69]. En esencia, un roguelike es un videojuego con pisos generados aleatoriamente, en el que el jugador irá obteniendo mejoras, pero a costa de aumentar la dificultad. Además, la muerte en este tipo de juegos suele ser permanente, por lo que cada partida es prácticamente única.
- **Sandbox RPG**: juegos con un mundo abierto enorme, en el que la libertad de movimiento y de acciones suelen ser parte esencial de dichos videojuego. *Fallout 4* [25] es un ejemplo de este tipo de videojuegos.

- **Videojuegos de simulación**: intentan ser lo más fieles posibles a la realidad en cuestión de manejar tanto los mandos de un avión, como una vida. Se pueden incluir los siguientes géneros en este tipo de videojuegos:

  - **Videojuegos de construcción y gestión**: videojuegos en los que la mecánica principal es gestionar y construir edificios, ciudades, tiendas, etc. El videojuego más representativo de este tipo es *SimCity* [75], en el que tendremos que gestionar nuestra ciudad.
  - **Simulador de vida**: juegos en los que tendremos que manejar una o más vidas artificiales. Pueden centrarse en las relaciones o pueden ser juegos de gestión de un ecosistema. La saga de *Los Sims* [47] es la más representativa de este género.
  - **Simuladores de conducción**: este tipo de videojuegos nos pone en los mandos de un vehículo, ya sea un tren, un coche o una nave espacial de la forma más realista posible. Un ejemplo de este tipo de videojuegos sería *Euro Truck Simulator* [24], que nos pone en los mandos de un camión para recorrer Europa entregando mercancía.

- **Videojuegos de estrategia**: centrados en mecánicas que necesitan pensar con cuidado cada acción que se realiza, y sobre todo, pensar a largo plazo, como en una partida de ajedrez.

- **Estrategia por turnos**: se refiere a juegos de estrategia en los que el jugador tiene un tiempo establecido para realizar acciones, normalmente controlando tropas, edificios, u otros tipos de entidades, mientras que el resto de los jugadores o la máquina tendrán que esperar a su turno. El referente de este género es, sin lugar a dudas, la saga *Sid Meier's Civilization* [16].
- **Estrategia en tiempo real (RTS)**: género en el que la mecánica principal se basa en obtener recursos y gestionarlos creando edificios, tropas y armas, igual que en la estrategia por turnos, pero con la diferencia de que todos los jugadores, sean estos, personas reales o controlados por el videojuego, juegan al mismo tiempo y por tanto deberán realizar decisiones constantemente. *Starcraft* [77] es uno de los videojuegos más jugados de este género, y su escena competitiva sigue viva a día de hoy.
- **Arena de batalla online multijugador (MOBA)**: género en el que un jugador controla a solo un personaje de uno de, normalmente, dos equipos. Usualmente, en este género encontramos estructuras definidas que reaparecen en cada partida. Además se generan periódicamente enemigos triviales, de manera que proporcionan recursos a los jugadores, que usarán para mejorar sus personajes y centrarse en el combate contra los jugadores del otro equipo. *League of Legends* [44] es uno de los MOBAs más jugados.
- **Tower defense**: es un género basado en la creación de estructuras que eliminarán a enemigos que usualmente van por un carril. Cuando un enemigo es eliminado otorga dinero para mejorar las estructuras o para crear nuevas. Un juego de este género sería *Orcs must die* [59].

- **Juegos de deportes**: videojuegos que se basan en algún deporte real, como puede ser el fútbol, el tenis, el baloncesto o incluso carreras de vehículos.

- **Juegos casuales y arcade**: juegos centrados en que las partidas duren poco, ya sea por el formato de partida rápida, como por ejemplo, jugada esperando el bus, o bien como los juegos casuales actuales y los arcade del

pasado, que se cobraba por partida. Los videojuegos casuales muchas veces beben de videojuegos arcade, ya tomando mecánicas de dichos videojuegos, o incluso tomando videojuegos enteros adaptándolos a móvil, como *Candy Crush* [14].

- **Idle games**: juegos cuya mecánica principal es una acción trivial, como comprar un edificio, y dejar el juego correr para que genere recursos y poder comprar más mejoras para obtener más recursos. El videojuego que mejor trata este tipo es *Cookie Clicker* [18] Este tipo de juego es el elegido para el videojuego que forma parte de este proyecto, *Pizza Clicker*.

Tengamos en cuenta, además, que diferentes géneros y tipos se pueden combinar en un solo videojuego para enriquecer las jugabilidades y, además, hacer avanzar el género. Por ejemplo, a un videojuego de simulación de vida podemos añadirle un sistema RPG para que cada vida que estemos manejando tenga ciertas habilidades que otros no, como sucede en diversos packs de expansiones de *Los Sims*.

2.3. Partes de un videojuego

Un videojuego, como cualquier problema, se puede subdividir en partes más pequeñas para entenderlo mejor, tanto si estamos analizando un producto ya realizado, como para hacer un desarrollo del mismo.

En este documento, para simplificar, se tratarán como partes de un videojuego la programación, el diseño de las mecánicas, también conocido como jugabilidad, y la historia. Además de esto, se considerará una cuarta parte, formada por el apartado estético o gráfico y por último el diseño de niveles.

2.3.1. Programación

La programación en un videojuego es una de las partes troncales, quizá la más importante, puesto que sin ella el resto del juego no podría funcionar.

Hay innumerables técnicas, prácticas y herramientas para realizar un videojuego, a más nivel o menos. Para este documento se cubrirán las diferencias que podemos encontrar entre programar directamente un videojuego desde cero, como se realizó en las asignaturas de videojuegos en el grado, y la programación usando un motor comercial, como puede ser Unity [86] o Unreal engine [88]. Ampliaremos sobre diferentes motores y alternativas en el capítulo 4, Análisis y especificación de sistemas, concretamente en el apartado de herramientas software.

Grosso modo, en los videojuegos realizados desde cero con gráficos de algún tipo, la programación se suele basar en un bucle de renderizado constante, para poder mostrar dichos gráficos y refrescarlos una vez por cada fotograma de juego, por lo que lo óptimo sería que dicho bucle se ejecutase más de 60 veces por segundo. Además de esto, deberíamos buscar o crear diferentes librerías para funciones básicas, como las físicas, la creación de eventos y exportación, entre otras.

En cambio, usando un motor comercial, tendremos muchísimas más ventajas que creando un juego desde cero, a pesar de perder el control sobre muchas de las partes del proceso de creación y de exportación. Entre dichas ventajas se destaca su relativa facilidad de uso que permite realizar prototipos de juegos de forma rápida y aumentar la productividad en el desarrollo de un videojuego [66]. Además de esto, evitamos reinventar la rueda en muchos casos, como por ejemplo puede ser la administración de archivos.

En el caso de este proyecto, se ha usado Unity, que se basa en programación orientada a objetos y a eventos [6]. Esto significa que, por ejemplo, se tendrá una clase jugador con todos los atributos que correspondan, entre ellos el modelo 3D o sprite (imagen en 2D) asociado que observará el jugador. Dicha clase

responderá a eventos, como por ejemplo, el pulsar la tecla de flecha derecha, de manera que se moverá hacia esa dirección.

2.3.2. Diseño de las mecánicas y jugabilidad

La jugabilidad de un videojuego está en su mayoría compuesta por mecánicas, siendo estas cualquier cosa que pueda significar una forma de interactuar con un videojuego. Ejemplos de esto podría ser el simple movimiento de Mario en *Super Mario Bros*, que sumado a la mecánica de salto, correr, aplastar enemigos y romper bloques daría como resultado la jugabilidad básica del juego.

Estas mecánicas pueden ser muy complejas de diseñar para que sean divertidas y gusten a los usuarios en un videojuego. Por ejemplo, un pequeño cambio en la velocidad de movimiento del citado Super Mario Bros puede hacer que el juego se sienta extremadamente rápido e injusto, o que sea una experiencia de juego tediosa, y por tanto, aburrida.

Además de esto, se debe tener en cuenta algo más, la kinestética, es decir, la relación entre el controlador y el comportamiento de las mecánicas. Esta puede cambiar dependiendo del juego que queramos hacer. Podemos hacer que nuestro personaje se mueva en el mismo instante que tocamos el botón, como podría suceder en un juego plataformas, como Super Mario Bros, o que al darle al mismo botón se inicie el movimiento, siendo este un poco más lento, para dar la sensación de que el personaje tiene “peso”, como sucede en *Grand Theft Auto (GTA) V* [32].

Por tanto las mecánicas deben estar ajustadas al milímetro, realizar testeos constantes y realizar pruebas con usuarios que no hayan tenido contacto previo con el juego antes de lanzar el mismo. Este proyecto puede resultar extremadamente útil en ese sentido. La capacidad de tener un feedback tan inmediato es extremadamente valiosa en esta industria.

2.3.3. Historia

Se podría decir que todo lo que nos podemos encontrar que provoque alguna reacción sentimental en nosotros tiene una historia, y los videojuegos no iban a ser menos.

Estas sensaciones las podemos observar en cualquier videojuego, como puede ser la felicidad de superar un nivel del mítico *Tetris* [82], la competitividad que puede generar que nos elimine otro jugador en *Overwatch,* o la nostalgia que puede evocar en nosotros *Undertale* [85]. Ahora bien, estas sensaciones pueden ser provocadas por diferentes aspectos de un videojuego, siendo estos principalmente la narrativa, es decir, los sucesos que podemos encontrar en un videojuego, y las mecánicas antes citadas. Estas dos partes deberían "contar la misma historia", es decir, no contradecirse la una a la otra, si no queremos encontrarnos con un caso de disonancia ludonarrativa [17], como por ejemplo, contar los horrores de la guerra desde la narrativa, mientras las mecánicas proporcionan unas sensaciones muy agradables al disparar, y recompensan al jugador por cada soldado despachado.

2.3.4. Estética

Podríamos definir la estética en un videojuego como el conjunto visual general que encontramos durante el desarrollo del mismo. Estaría compuesto por todos los elementos visuales, entre ellos texturas, modelos, aspecto de la luz, posicionamiento de la cámara para un videojuego con gráficos tridimensionales, o los sprites en un juego de dos dimensiones.

Esta estética puede cambiar tremendamente de un videojuego a otro, reforzando el resto de apartados de un videojuego, especialmente la historia del mismo. Así pues, podemos encontrar grandes diferencias, incluso dentro de géneros de videojuegos con el mismo motor y con ideas similares. Por ejemplo, *Hollow Knight* y *Guacamelee* responden al género metroidvania, compartiendo también el mismo tipo de gráficos, bidimensionales, pero tienen una estética radicalmente diferente, como se puede observar en la Figura 3 y en la Figura 4, respectivamente.

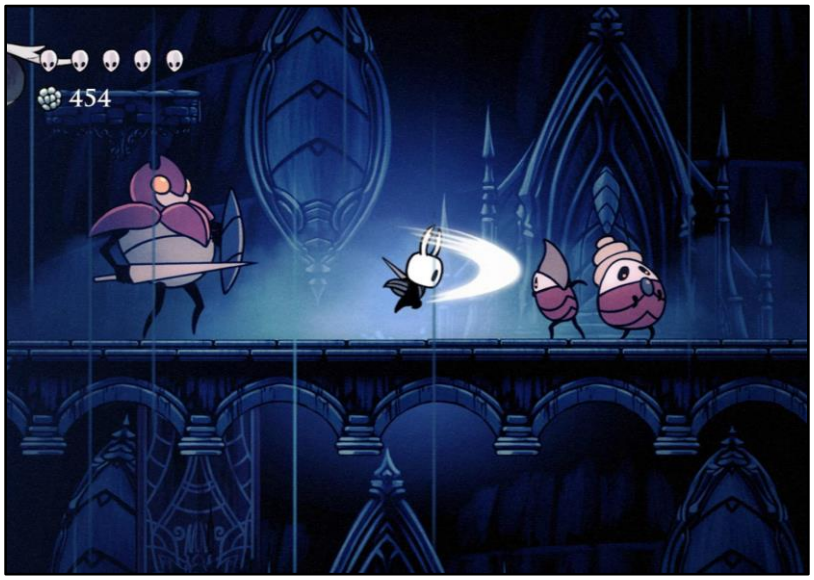

Figura 3: *Hollow Knight.*

Fuente: Kit de prensa en hollowknight.com.

Figura 4: *Guacamelee.*

Fuente: Steam https://store.steampowered.com/app/275390.

2.3.5. Diseño de niveles

En un videojuego nos podemos encontrar con diferentes situaciones que raramente se podrían dar en la vida real respecto al entorno. Por ejemplo, no nos encontramos, casualmente, medios tubos por la calle cada pocos metros, como ocurre en los videojuegos de skate, ni en un conflicto armado ambos frentes están diseñados para estar equilibrados en posibilidades de victoria o derrota.

Todo esto es posible en los videojuegos gracias al diseño de niveles, o lo que es lo mismo, "la creación de niveles, misiones, mapas, entornos de videojuegos, pantallas y cualquier otro espacio donde el jugador o su avatar interaccionan con el mundo del videojuego" [74], y estas diferencias respecto a la vida real se dan porque no sería divertido un juego de skate en el que no puedas patinar, o un juego de disparos multijugador en el que, en cierto mapa, tus posibilidades de victoria fueran nulas.

Por tanto, podemos discernir que, el diseño de niveles de un videojuego es otra de las partes importantes de un videojuego, y que tener conocimientos sobre cómo lo están interpretando los jugadores no solamente es importante, sino crítico.

2.4. Métricas

Los diferentes tipos de métricas, así como sus representaciones, son claves en este proyecto. Por lo tanto, ha habido que barajar cuáles de estas métricas son importantes.

En primer lugar nos encontramos los DAU (Daily Active Users), o los usuarios activos diarios, es decir, la cantidad de usuarios que entran al videojuego diariamente. Esta métrica es importante para ver cómo evoluciona un videojuego en el tiempo. Está íntimamente relacionada con los MAU (Monthly Active Users), usuarios activos mensuales [37].

Dividiendo los usuarios activos diarios por los mensuales obtenemos una relación interesante, ya que nos permite obtener el porcentaje de usuarios que se han quedado jugando durante el mes.

También hay que considerar los beneficios de dichos juegos, por lo menos en el entorno móvil, por lo que se ha de tener en cuenta la tasa de conversión, es decir, el porcentaje de usuarios que ha realizado una compra respecto al total de usuarios. Habría que observar, además, los ingresos diarios del juego durante el tiempo.

Por último estarían las métricas en las que se centra este proyecto, es decir, las métricas referentes al diseño de los videojuegos, ya sea por el diseño de niveles o por la accesibilidad del mismo. Métricas que podrían ayudar a mejorar estos apartados serían, por ejemplo, ver cuánto tiempo tarda un usuario en darle a un botón adecuado en una interfaz, ver el tiempo medio por usuario gastado en los menús, o ver las muertes que se dan en cierto punto de un nivel.

Estas métricas son muy importantes para que el diseño de los videojuegos coincida con los resultados observados por los jugadores. Por ejemplo, supongamos que en el diseño se ha decidido que el nivel 2 va a ser un pequeño desafío, pero que no debe ser solventado con más de 3 intentos de media por usuario. Sin embargo, al analizar las métricas se percibe que los usuarios raramente llegan al siguiente nivel y, en el caso de llegar, mueren de media unas 8 veces. Ante esto deberíamos pensar en realizar alguna modificación en el nivel, con el objetivo de simplificarlo para que el diseño coincida con el producto deseado. Sin este análisis, solo se podría haber corregido posteriormente con las opiniones de los usuarios, si se tiene la suerte de tenerlas.

2.5. Breve historia y actualidad de la industria

La industria del videojuego ha cambiado mucho en su corta vida. Inicialmente, los videojuegos solo estaban disponibles en las máquinas arcade que, previo pago de una moneda de 100 pesetas, proporcionaban una partida a juegos como *Metal Slug* [50] o *Snow Bros* [76], hasta que aparecieron las consolas. Estos dispositivos permitían pagar una única vez por todas las partidas que se quisiera, y por tanto, poco a poco, desbancaron a las máquinas recreativas.

Pero ¿Qué hay de la actualidad? ¿Siguen las consolas reinando? La realidad, aunque le pese a ciertos sectores de la industria, es que las videoconsolas siguen reinando [10], pero más sofisticadas y con más métodos de monetización, como por ejemplo los DLCs, contenido descargable para los videojuegos, o los micropagos, por ejemplo, comprar vidas en *Candy Crush* [14].

Respecto a estos últimos, los micropagos, se ha generado la última tendencia en el desarrollo de videojuegos, y es hacia donde va ahora mismo la industria, que es el videojuego como servicio. Juegos que no tienen fin, que siempre quieren que vuelvas y que, a ser posible, hagas alguna pequeña compra. Esto se realiza tanto con videojuegos con coste habitual, como puede ser 70 euros actualmente, como con videojuegos gratuitos, como los que podemos encontrar en la tienda de aplicaciones de Google o la de iOS.

Se debe tener en cuenta también que las diferencias de desarrollo entre un videojuego en sus albores y en la actualidad son abismales, tanto por complejidad, como por coste, y que por tanto, realizar videojuegos de gran calidad para un solo jugador, títulos que se venden menos actualmente, es prácticamente el mayor riesgo que se puede cometer en la industria. Confirma esto que, según datos recogidos por Wikipedia [45], de entre los diez primeros de la lista de videojuegos más costosos de desarrollar, solamente se encuentra uno anterior a la generación de Xbox 360, y Playstation 3, en concreto *Final Fantasy VII* [26], videojuego de rol archiconocido y el favorito de la gran mayoría de jugadores que se inició en los videojuegos con la PlayStation 1 (Sony, 1999).

El problema es que para alcanzar una posición estable, es decir, que un videojuego ingrese más de lo que cuesta en todos los estados del desarrollo, con un videojuego con este enfoque es necesario intentar tener en cuenta todas las variables posibles y no dejar nada al azar. Elegir bien qué tipo de modelo de negocio se implementará, qué coste tendrá y que el juego en sí sea especialmente atrayente, para que aumente el alcance del mismo, son factores clave para conseguir que un videojuego genere beneficios. Por tanto, en ese tipo de desarrollos, las métricas, como las que proporcionará este proyecto, son casi obligadas si se requiere tener un mínimo éxito que pueda cubrir los costes.

3. Justificación y objetivos

Los videojuegos han sido una revolución en el sector del ocio en los últimos 30 años, moviendo capital, atrayendo inversores y divirtiendo allá donde van. Sabiendo esto y teniendo en cuenta el estado actual de la industria del videojuego y su oferta, los videojuegos deben ser cada vez mejores y, de hecho, los jugadores cada vez exigen más características a los mismos.

Este proyecto tiene como objetivo exactamente eso, intentar ayudar a mejorar los videojuegos que lo utilicen, mediante el envío de datos directamente desde el videojuego al sistema, analizando dichos datos mediantes métricas, conociendo así qué puntos fuertes y débiles tienen nuestros videojuegos. Además de esto podremos conocer dónde hay una disonancia entre la concepción de un sistema y el resultado actual provocado por los jugadores, como podría ser la dificultad de una zona planeada como sencilla, pero que muchos jugadores sufren para superar.

Este objetivo no sería posible si no tuviera el conocimiento obtenido en el grado de Ingeniería Multimedia de la Universidad de Alicante, cuyos objetivos son compartidos en gran manera con los de este proyecto. Además, este proyecto utiliza conocimientos que solo pueden ser extraídos de los dos itinerarios del grado. Por tanto, atendiendo a que realicé asignaturas de ambos itinerarios, el proyecto se ha podido llevar a buen término.

El primero de dichos itinerarios es el de “gestión de contenidos”, basado en la gestión de contenidos multimedia, que nos servirá de mucha ayuda para poder administrar la gran cantidad de datos que generará este proyecto. Además de este itinerario, en Ingeniería Multimedia encontramos el de “creación de entretenimiento digital”, que está principalmente enfocado a la realización de videojuegos.

Teniendo en cuenta mi participación en ambos itinerarios, y los temarios, competencias y objetivos de Ingeniería Multimedia, además de la necesidad de poder tener el control sobre muchos detalles en los videojuegos, se llegó a la

conclusión de que este proyecto sería un buen trabajo de fin de grado, puesto que este debe ser una prueba tácita de que los conocimientos adquiridos durante la carrera han calado y permiten al ingeniero multimedia ser capaz de manejar sistemas como el que se propone aquí.

Concretamente, el objetivo de “Videogames for a living” es conseguir tener un sistema capaz de analizar cualquier aspecto que deseemos de un videojuego. Para ello se plantea diseñar un sistema que pueda recibir datos desde un videojuego, que los formatee y que los pueda mostrar según lo especifique un usuario, que en este caso sería un desarrollador de videojuegos. Los objetivos específicos del proyecto serían los siguientes:

- Tener un sistema capaz de almacenar datos de múltiples tipos organizado con una base de datos.
- Crear un servidor en el que se almacenará la lógica interna e interactúe con la base de datos.
- Que dicho sistema pueda enviar y recibir dichos datos mediante peticiones HTTP, por tanto se creará una API Restful.
- Permitir que se creen, editen y borren diferentes juegos.
- Que cada juego tenga sus métricas y variables, y que estas se puedan crear, editar y borrar.
- Mostrar dichas métricas en una página web en métricas limpias y fáciles de interpretar por el usuario.
- Permitir al usuario enviar datos desde un videojuego mediante petición HTTP.
- Crear un videojuego “infinite runner”.
- Crear un clicker.
- Enviar datos desde los dos juegos.
- Que dichos videojuegos sean jugables y divertidos en su medida.

4. Análisis y especificación de sistemas

El análisis de requisitos y la especificación del sistema es clave en el desarrollo de un sistema multimedia, como es este. Del análisis y la especificación dependen el resto de partes del desarrollo, además de apuntar una hoja de ruta y servir para estimar y conocer los riesgos y costes que podemos encontrar en este proyecto.

A lo largo de este capítulo se especificarán los diferentes aspectos del análisis, con el objetivo de tenerlos detallados y que sirvan de guía para el resto del desarrollo. Estos aspectos son los siguientes:

- Captura de requisitos, es decir, la especificación de todas las funciones que debe tener nuestro sistema, de forma que se tengan como referencia durante todo el desarrollo. Diferenciaremos entre requisitos funcionales, es decir, requisitos que constituirán una función del sistema, y requisitos no funcionales, que no constituyen una característica del sistema, pero que serán necesarios para que el sistema funcione. Los requisitos de este proyecto se detallan en la Sección 4.1.
- Casos de uso, las acciones que serán capaces de hacer los principales usuarios del sistema. Se desarrollarán los casos de uso de este proyecto en la Sección 4.2.
- Herramientas que vamos a utilizar para realizar el proyecto, siendo estas software y hardware en su totalidad. Las herramientas usadas en este proyecto serán expuestas y comparadas con alternativas en la Sección 4.3.
- Ámbito de uso y público objetivo, es decir, a quién va dirigido y en qué ámbitos se puede usar. Se tratarán estos aspectos en la Sección 4.4.
- Metodología utilizada, que se explicará en la Sección 4.5
- Planificación del proyecto, analizando las distintas tareas que han de realizarse y estimando el tiempo necesario para llevarlo a buen término. Estos aspectos se tratarán en la Sección 4.6.

4.1. Captura de requisitos

Los requisitos son las características que debemos encontrar en nuestro sistema. Como se ha comentado anteriormente, estos requisitos se pueden dividir en dos, requisitos funcionales, es decir, acciones que podrá realizar el usuario o que realizará el sistema, y requisitos no funcionales, requisitos que no representarán ninguna acción del sistema o del usuario, pero que son necesarios para poder realizar requisitos funcionales. Para una mejor organización de los requisitos se clasificarán usando tablas, que tendrán las siguientes partes:

- Identificador único que nos permitirá no confundir en ningún momento un requisito con otro. Estará compuesto por el prefijo RF (requisito funcional) o RNF (requisito no funcional), seguido de a qué parte del proyecto se refiere y el número de requisito en dicha parte del proyecto.
- Prioridad, utilizando el método MoSCoW [51], una técnica de priorización usada en el desarrollo de software basada en que todos los requisitos tendrán una letra asignada:
    - M para Must have, lo que significa que dicha funcionalidad es indispensable para el funcionamiento del sistema.
    - S para Should have, requisitos que, si bien el sistema podrá funcionar sin ellos, son de gran importancia.
    - C para Could have, requisitos que son deseables, pero no necesarios, principalmente mejoras de la experiencia de usuario.
    - W para Won't have o Would have, los requisitos menos críticos, o incluso utópicos.
- Título, definición en menos de 3 palabras de lo que va a ser el requisito en cuestión.
- Relaciones que encontraremos entre requisitos, como relaciones de dependencia, o concatenaciones de requisitos.

4.1.1. Requisitos funcionales

A continuación se definen los requisitos funcionales del sistema (véase Tabla 1-Tabla 19):

4.1.1.1. Página web

Identificador	RF-PW-01
Prioridad	M
Título	Crear juego.
Descripción	El desarrollador podrá crear un juego.
Relaciones	

Tabla 1: *Requisito funcional RF-PW-01. Fuente: Elaboración Propia.*

Identificador	RF-PW-02
Prioridad	M
Título	Añadir, editar, borrar variables.
Descripción	El desarrollador podrá crear, borrar o editar una variable con nombre.
Relaciones	Necesitaríamos tener creado un juego con RF-PW-01.

Tabla 2: *Requisito funcional RF-PW-02. Fuente: Elaboración Propia.*

Identificador	RF-PW-03
Prioridad	M
Título	Crear métrica.
Descripción	El desarrollador deberá poder crear métricas, con nombre, tipo de métrica y qué variables se seguirán.
Relaciones	Necesitaríamos tener creado un juego con RF-PW-01.

Tabla 3: *Requisito funcional RF-PW-03. Fuente: Elaboración Propia.*

Identificador	RF-PW-04
Prioridad	M
Título	Ver métricas realizadas.
Descripción	El desarrollador deberá poder ver las diferentes métricas que habrá creado.
Relaciones	Necesitaríamos haber realizado RF-PW-02 y RF-PW-03.

Tabla 4: *Requisito funcional RF-PW-04. Fuente: Elaboración Propia.*

Identificador	RF-PW-05
Prioridad	S
Título	Seleccionar diferentes juegos.
Descripción	Cada desarrollador podrá tener varios juegos, y por tanto seleccionar sobre cuál necesita ver métricas.
Relaciones	Necesitaríamos tener creado un juego con RF-PW-01.

Tabla 5: *Requisito funcional RF-PW-05. Fuente: Elaboración Propia.*

Identificador	RF-PW-06
Prioridad	S
Título	Cambiar nombre e imagen de juego.
Descripción	Un desarrollador debe poder editar los datos a mostrar de un juego, como su nombre o su imagen.
Relaciones	Necesitaríamos haber realizado RF-PW-01.

Tabla 6: *Requisito funcional RF-PW-06. Fuente: Elaboración Propia.*

Identificador	RF-PW-07
Prioridad	S
Título	Borrar juego.
Descripción	Un desarrollador debe poder borrar un juego.
Relaciones	Necesitaríamos haber realizado RF-PW-01.

Tabla 7: *Requisito funcional RF-PW-07. Fuente: Elaboración Propia.*

Identificador	RF-PW-08
Prioridad	C
Título	Comparar métricas.
Descripción	Un desarrollador debe poder comparar métricas, tanto del juego que estemos viendo, como de otros juegos de ese desarrollador.
Relaciones	Necesitaríamos haber realizado RF-PW-03.

Tabla 8: *Requisito funcional RF-PW-08. Fuente: Elaboración Propia.*

Identificador	RF-PW-09
Prioridad	C
Título	Editar métricas.
Descripción	Un desarrollador debe poder cambiar cualquier aspecto de una métrica.
Relaciones	Necesitaríamos haber realizado RF-PW-03.

Tabla 9: *Requisito funcional RF-PW-09. Fuente: Elaboración Propia.*

Identificador	RF-PW-10
Prioridad	M
Título	Borrar métricas.
Descripción	Un desarrollador debe poder borrar una métrica.
Relaciones	Necesitaríamos haber realizado RF-PW-03.

Tabla 10: *Requisito funcional RF-PW-10. Fuente: Elaboración Propia.*

4.1.1.2. Penguin Rush

Identificador	RF-PR-01
Prioridad	M
Título	Iniciar partida.
Descripción	El sistema permitirá al jugador iniciar partida.
Relaciones	

Tabla 11: *Requisito funcional RF-PR-01. Fuente: Elaboración Propia.*

Identificador	RF-PR-02
Prioridad	M
Título	Moverse.
Descripción	El sistema permitirá al jugador mover el personaje principal, en este caso un pingüino.
Relaciones	

Tabla 12: *Requisito funcional RF-PR-02. Fuente: Elaboración Propia.*

Identificador	RF-PR-03
Prioridad	M
Título	Perder.
Descripción	El sistema tendrá un estado de derrota.
Relaciones	

Tabla 13: *Requisito funcional RF-PR-03. Fuente: Elaboración Propia.*

Identificador	RF-PR-04
Prioridad	S
Título	Acumular puntos.
Descripción	El sistema tendrá un sistema de acumulación de puntos para poder ver como se ha hecho respecto a partidas anteriores.
Relaciones	

Tabla 14: *Requisito funcional RF-PR-04. Fuente: Elaboración Propia.*

Identificador	RF-PR-05
Prioridad	M
Título	Enviar variables al servidor.
Descripción	El desarrollador debe poder seleccionar una variable creada para ligar al sistema. Esta variable beberá de las múltiples instancias de la misma.
Relaciones	Necesitaríamos tener creado un juego, y una variable con RF-PW-02.

Tabla 15: *Requisito funcional RF-PR-05. Fuente: Elaboración Propia.*

4.1.1.3. Pizza Clicker

Identificador	RF-PC-01
Prioridad	S
Título	Elegir ingredientes.
Descripción	El jugador podrá elegir diferentes ingredientes para realizar su pizza.
Relaciones	

Tabla 16: *Requisito funcional RF-PC-01. Fuente: Elaboración Propia.*

Identificador	RF-PC-02
Prioridad	S
Título	Vender pizza.
Descripción	El jugador podrá vender la pizza que ha hecho para aumentar su dinero y el número de pizzas vendidas.
Relaciones	El jugador deberá haber elegido los ingredientes en RF-PC-01.

Tabla 17: *Requisito funcional RF-PC-02. Fuente: Elaboración Propia.*

Identificador	RF-PC-03
Prioridad	S
Título	Comprar mejoras.
Descripción	El jugador podrá comprar mejoras que le ayudarán a hacer pizzas más rápido.
Relaciones	El jugador deberá tener dinero, que consiguió en RF-PC-02.

Tabla 18: *Requisito funcional RF-PC-03. Fuente: Elaboración Propia.*

Identificador	RF-PC-04
Prioridad	M
Título	Enviar variables al servidor.
Descripción	El desarrollador debe poder seleccionar una variable creada para ligar al sistema. Esta variable beberá de las múltiples instancias de la misma.
Relaciones	Necesitaríamos tener creado un juego, y una variable con RF-PW-02.

Tabla 19: *Requisito funcional RF-PC-04. Fuente: Elaboración Propia.*

4.1.2. Requisitos no funcionales

A continuación se definen los requisitos no funcionales del sistema (véase Tabla 20-Tabla 24):

4.1.2.1. Página web

Identificador	RNF-PW-01
Prioridad	M
Título	Recibir información de videojuego.
Descripción	El sistema recibirá por petición HTTP los datos del desarrollo de videojuegos en formato JSON (*JavaScript Object Notation*).
Relaciones	

Tabla 20: *Requisito no funcional RNF-PW-01. Fuente: Elaboración Propia.*

Identificador	RNF-PW-02
Prioridad	M
Título	Formatear datos.
Descripción	El sistema formateará los datos para poder transformarlos en métricas.
Relaciones	Necesario RNF-PW-01.

Tabla 21: *Requisito no funcional RNF-PW-02. Fuente: Elaboración Propia.*

Identificador	RNF-PW-03
Prioridad	M
Título	Velocidad.
Descripción	El sistema no podrá tardar más de 2 segundos entre pantalla y pantalla.
Relaciones	

Tabla 22: *Requisito no funcional RNF-PW-03. Fuente: Elaboración Propia.*

4.1.2.2. Penguin Rush

Identificador	RNF-PR-01
Prioridad	M
Título	Velocidad.
Descripción	El sistema deberá moverse a una velocidad adecuada, es decir, a más de 30 fps.
Relaciones	

Tabla 23: *Requisito no funcional RNF-PR-01. Fuente: Elaboración Propia.*

4.1.2.3. Pizza Clicker

Identificador	RNF-PC-01
Prioridad	M
Título	Velocidad.
Descripción	El sistema deberá moverse a una velocidad adecuada, es decir, a más de 30 fps.
Relaciones	

Tabla 24: *Requisito no funcional RNF-PC-01. Fuente: Elaboración Propia.*

4.2. Casos de uso

Los casos de uso son descripciones de las diferentes acciones que podrán realizar los diferentes actores que encontraremos en un sistema. Estos actores pueden ser, tanto usuarios como desarrolladores o incluso sistemas.

En este proyecto tendremos en cuenta los casos de uso de la página web y de los videojuegos, por lo que tendremos entre nuestros actores el desarrollador de videojuegos que va a editar las métricas en la página web, y el jugador en los videojuegos. Se definirán los casos de uso con la siguiente estructura:

- Identificador del caso de uso: un identificador único para cada caso de uso compuesto por CU (caso de uso), las siglas de la parte a la que pertenece el caso de uso, y el número de caso de uso.
- Nombre del caso de uso: un título explicativo para un caso de uso.
- Nombre del actor: nombre del agente que realizará la acción del caso de uso.
- Descripción del caso de uso: se describirá en esta parte el caso de uso como tal.
- Requisitos: requisitos necesarios para poder realizar el correspondiente caso de uso.

4.2.1. Página web

El diagrama de casos de uso de la parte de la página web se resume en la Figura 5.

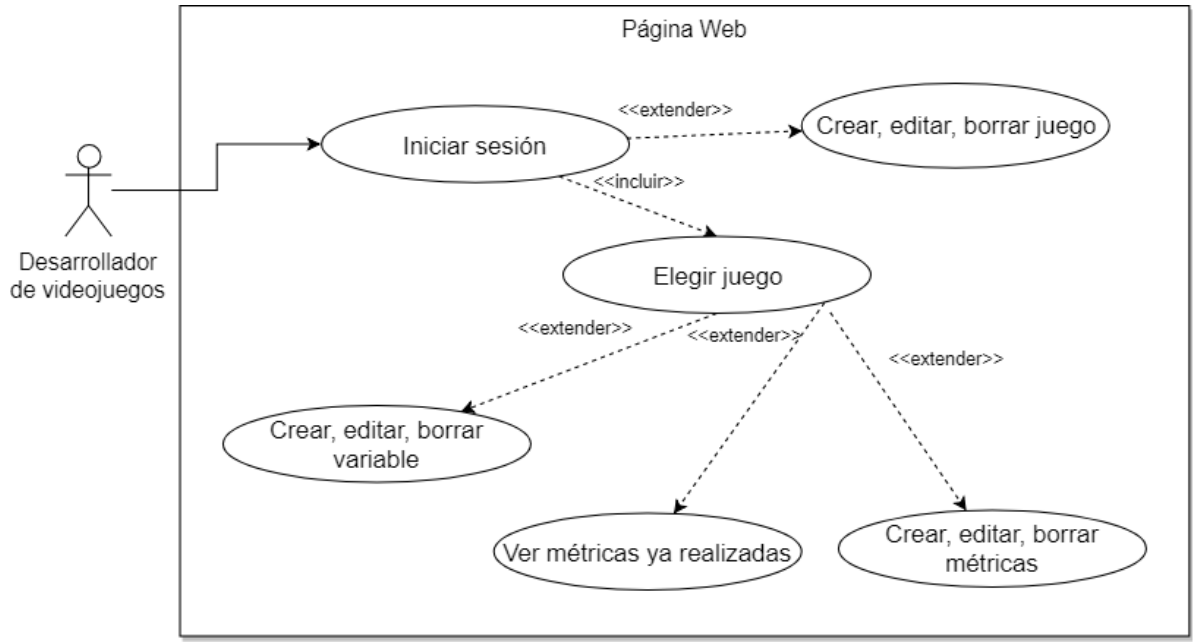

Figura 5: *Casos de uso de página web.*

Fuente: Elaboración propia.

Las definiciones de los casos de uso vienen esquematizados en las siguientes tablas (véase Tabla 25-Tabla 36):

Identificador	CU-PW-01
Caso de uso	Crear juego.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá crear juegos.
Requisitos	

Tabla 25: *Caso de uso CU-PW-01. Fuente: Elaboración Propia.*

Identificador	CU-PW-02
Caso de uso	Elegir juego.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador elegirá el juego que desee.
Requisitos	CU-PW-01.

Tabla 26: *Caso de uso CU-PW-02. Fuente: Elaboración Propia.*

Identificador	CU-PW-03
Caso de uso	Borrar juego.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá elegir borrar un juego.
Requisitos	CU-PW-01.

Tabla 27: *Caso de uso CU-PW-03. Fuente: Elaboración Propia.*

Identificador	CU-PW-04
Caso de uso	Editar juego.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá elegir cambiar el nombre y la imagen de un juego.
Requisitos	CU-PW-01.

Tabla 28: *Caso de uso CU-PW-04. Fuente: Elaboración Propia.*

Identificador	CU-PW-05
Caso de uso	Crear métrica.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá crear una métrica, poniéndole nombre y eligiendo qué tipo de métrica es.
Requisitos	CU-PW-01.

Tabla 29: *Caso de uso CU-PW-05. Fuente: Elaboración Propia.*

Identificador	CU-PW-06
Caso de uso	Borrar métrica.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador puede borrar una métrica.
Requisitos	CU-PW-05.

Tabla 30: *Caso de uso CU-PW-06. Fuente: Elaboración Propia.*

Identificador	CU-PW-07
Caso de uso	Editar métrica.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador puede editar una métrica.
Requisitos	CU-PW-06.

Tabla 31: *Caso de uso CU-PW-07. Fuente: Elaboración Propia.*

Identificador	CU-PW-08
Caso de uso	Ver métricas.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá ver las métricas que ha guardado.
Requisitos	CU-PW-06.

Tabla 32: *Caso de uso CU-PW-08. Fuente: Elaboración Propia.*

Identificador	CU-PW-09
Caso de uso	Añadir variable.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá añadir una variable a un juego.
Requisitos	CU-PW-01.

Tabla 33: *Caso de uso CU-PW-09. Fuente: Elaboración Propia.*

Identificador	CU-PW-10
Caso de uso	Borrar variable.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá borrar una variable.
Requisitos	CU-PW-09.

Tabla 34: *Caso de uso CU-PW-10. Fuente: Elaboración Propia.*

Identificador	CU-PW-11
Caso de uso	Editar variable.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá editar una variable, cambiando el nombre.
Requisitos	CU-PW-09.

Tabla 35: *Caso de uso CU-PW-11. Fuente: Elaboración Propia.*

Identificador	CU-PW-12
Caso de uso	Añadir variable a métrica.
Actor	Desarrollador de videojuegos.
Descripción	El desarrollador podrá añadir una variable a una métrica.
Requisitos	CU-PW-05 y CU-PW-09.

Tabla 36: *Caso de uso CU-PW-12. Fuente: Elaboración Propia.*

4.2.2. Penguin Rush

El diagrama de casos de uso del videojuego *Penguin Rush* se resume en la Figura 6.

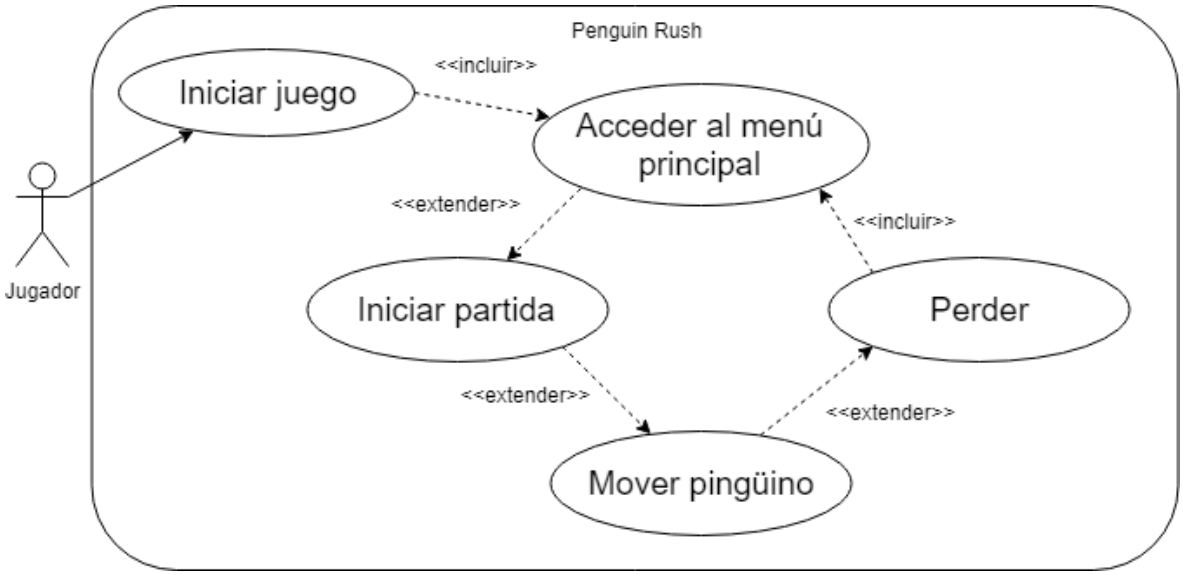

Figura 6: *Casos de uso de Penguin Rush. Fuente: Elaboración Propia.*

La definición de los casos de uso viene esquematizada en las siguientes tablas (Tabla 37-Tabla 40):

Identificador	CU-PR-01
Caso de uso	Iniciar juego.
Actor	Jugador.
Descripción	Se inicia el juego.
Requisitos	

Tabla 37: *Caso de uso CU-PR-01. Fuente: Elaboración Propia.*

Identificador	CU-PR-02
Caso de uso	Iniciar partida.
Actor	Jugador.
Descripción	Se inicia la partida.
Requisitos	CU-PR-01.

Tabla 38: *Caso de uso CU-PR-02. Fuente: Elaboración Propia.*

Identificador	CU-PR-03
Caso de uso	Mover pingüino.
Actor	Jugador.
Descripción	El jugador puede mover el pingüino de izquierda a derecha tocando el lado opuesto de la pantalla respecto a la posición del pingüino.
Requisitos	CU-PR-02.

Tabla 39: *Caso de uso CU-PR-03. Fuente: Elaboración Propia.*

Identificador	CU-PR-04
Caso de uso	Perder.
Actor	Jugador.
Descripción	Si el jugador cae del tobogán, entonces perderá y aparecerá en el menú principal con los puntos que ha conseguido.
Requisitos	CU-PR-03.

Tabla 40: *Caso de uso CU-PR-04. Fuente: Elaboración Propia.*

4.2.3. Pizza Clicker

El diagrama de casos de uso del videojuego *Pizza Clicker* se resume en la Figura 7.

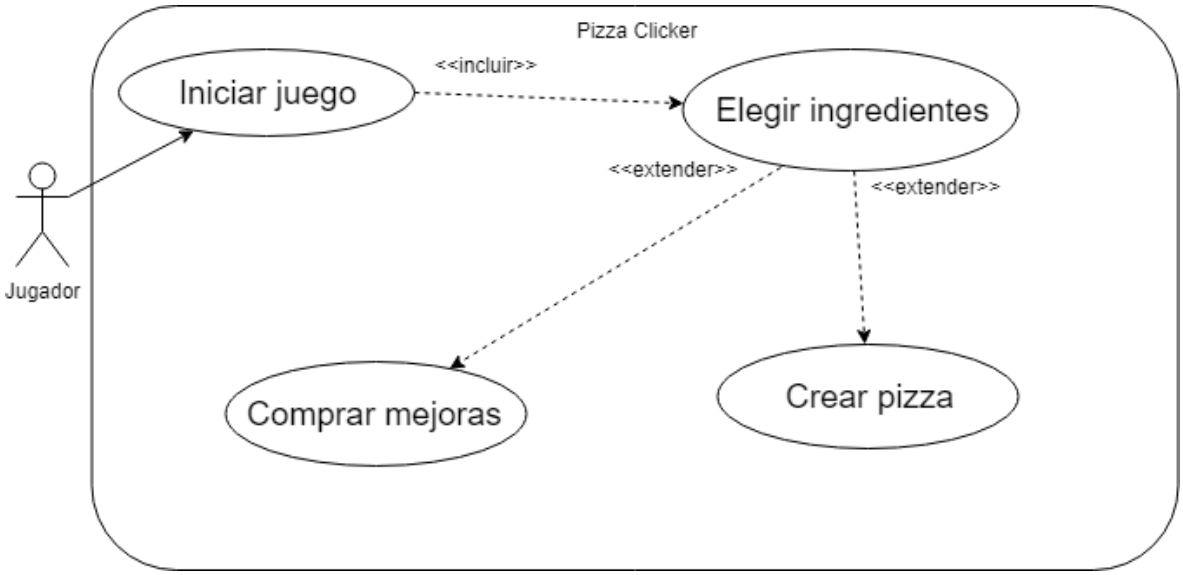

Figura 7: *Casos de uso de Pizza Clicker.*
Fuente: Elaboración Propia.

La definición de los casos de uso viene esquematizada en las siguientes tablas (Tabla 41-Tabla 44):

Identificador	CU-PC-01
Caso de uso	Iniciar juego.
Actor	Jugador.
Descripción	Se inicia el juego.
Requisitos	

Tabla 41: *Caso de uso CU-PC-01. Fuente: Elaboración Propia.*

Identificador	CU-PC-02
Caso de uso	Elegir ingredientes.
Actor	Jugador.
Descripción	El jugador puede elegir diferentes ingredientes para crear su pizza.
Requisitos	CU-PC-01

Tabla 42: *Caso de uso CU-PC-02. Fuente: Elaboración Propia.*

Identificador	CU-PC-03
Caso de uso	Crear pizza.
Actor	Jugador.
Descripción	El jugador podrá crear la pizza cuando esté de acuerdo con los ingredientes, y aumentar su dinero y pizzas realizadas.
Requisitos	CU-PC-02.

Tabla 43: *Caso de uso CU-PC-03. Fuente: Elaboración Propia.*

Identificador	CU-PC-04
Caso de uso	Comprar mejoras.
Actor	Jugador.
Descripción	El jugador puede comprar mejoras de manera que se puedan crear pizzas más rápido.
Requisitos	Tener dinero conseguido en CU-PC-03.

Tabla 44: *Caso de uso CU-PC-04. Fuente: Elaboración Propia.*

4.3. Herramientas software y hardware

Para el desarrollo de este sistema se han usado diversas herramientas, tanto hardware como software. En el caso del software, en la mayoría de casos se ha intentado usar software libre o, en algunos casos, gratuito para estudiantes.

4.3.1. Herramientas software

Entre las herramientas de software elegidas para este proyecto aparecen las típicas herramientas para un proyecto web con tal persistencia, además de las herramientas usadas para el desarrollo de videojuegos.

En particular, para el desarrollo web, necesitaremos entre otras muchas cosas, un servidor en el que almacenar la lógica de nuestro proyecto, una base de datos que interactúe con dicho servidor, y una página web que pueda interactuar con el mismo servidor.

Para el desarrollo de los videojuegos, necesitaremos un motor gráfico multiplataforma, para poder establecer las “normas” del videojuego, además de un programa que nos permita crear nuestros propios modelos 3D.

Además, deberemos tener en cuenta algunas herramientas extras que se usarán para ambos desarrollos, siendo estas un programa de edición de imagen, tanto para texturas de los modelos del videojuego como para imágenes que puedan ser usadas en el diseño web, además de para el desarrollo de este documento, y un sistema de control de versiones y almacenamiento del código, que nos permitirá almacenar los cambios en diferentes iteraciones para poder acceder fácilmente a un estado del código anterior.

Por último, necesitaremos otras herramientas para el diseño y la elaboración de este documento, como pueden ser un editor de texto, un programa que permita realizar diagramas UML (Lenguaje Unificado de Modelado) [43] fácilmente, como los vistos anteriormente, y un sistema de almacenamiento para todos estos recursos en la nube.

Muchas de las elecciones realizadas podrían sustituirse por otras alternativas, pero se ha dado prioridad a las nuevas tecnologías y al software abierto. Concretamente, las herramientas usadas a nivel de software han sido las siguientes:

- Desarrollo de videojuegos:
  - Unity 2018 [86] como motor principal.
  - Blender 3D [12] para editar y crear modelos.
- Sistema web:
  - Node.js [58] para la parte de servidor.
  - Angular 6 [4] para el frontend.
  - Bootstrap 4 [13] para el diseño del frontend.
  - MySQL [57] para la persistencia de datos.
  - Algunos módulos de Node.js.
- Ambos:
  - GitHub [30] para llevar el control de las versiones.
  - Photoshop [63] para editar y crear elementos introducidos en el videojuego o assets.
- Documento:
  - Documentos de Google Drive y Microsoft Word como procesadores de texto.
  - Google Drive como almacenamiento en la nube de diversos recursos para el documento, así como para el documento mismo.

4.3.1.1. Herramientas de desarrollo de videojuegos, Unity

Para el desarrollo de los videojuegos se ha usado Unity 2018 [86] como motor gráfico, ya que las alternativas, que luego serán expuestas, no me parecieron del todo adecuadas o suficientemente sencillas para el proyecto actual. Unity es un motor gráfico desarrollado por Unity Technologies, que entre sus características están las principales razones por la que ha sido elegido para este proyecto, es gratuito (mientras no factures más de 100.000 dólares con un videojuego hecho con Unity), sencillo de usar, con su modelo orientado a eventos y a objetos ya implementado, y es multiplataforma, de manera que podremos crear videojuegos para móvil, PC, o consola de una manera sencilla.

Este motor ha sido usado por una cantidad muy alta de videojuegos, algunos de ellos con millones de jugadores simultáneos, como *Hearthstone* [36], otros con un apartado artístico excepcional, como *Cuphead* [19] y *Ori and the Blind Forest* [60] tal como puede apreciarse en la Figura 8 y en la Figura 9, respectivamente.

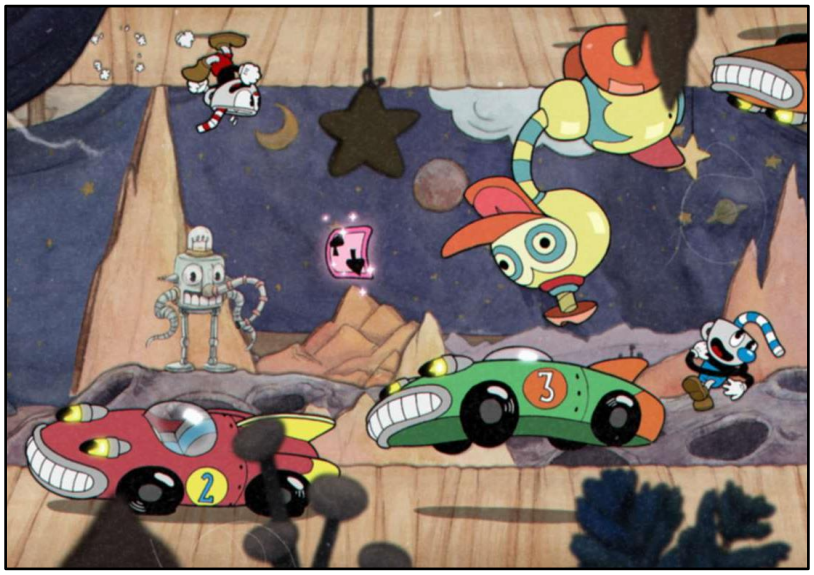

Figura 8: *Cuphead.*

Fuente: Tienda Microsoft, https://www.microsoft.com/es-es/p/cuphead/9njrx71m5x9p#.

Figura 9: *Ori and the Blind Forest.*

Fuente: Sección Media en https://www.orithegame.com.

Las alternativas que se han barajado junto a Unity han sido las siguientes:

- GameMaker [28]: es una herramienta de creación de videojuegos basada en un lenguaje de programación interpretado, esto quiere decir que, en lugar de compilar el código para traducir a lenguaje máquina, guardando en un archivo un ejecutable, como hacen Unity y Unreal Engine, es interpretado al mismo tiempo que se está ejecutando. Está orientado a programadores novatos, o incluso gente con conocimientos mínimos de programación. Aun así, esta herramienta ha sido usada para videojuegos muy interesantes, entre ellos, el ya mencionado en este documento, Undertale y el videojuego *Hyper Light Drifter* [39], cuya estética puede verse en la Figura 10 y en la Figura 11, respectivamente. Se descartó por su excesiva sencillez, además de su coste de cien dólares para publicar juegos en Android.

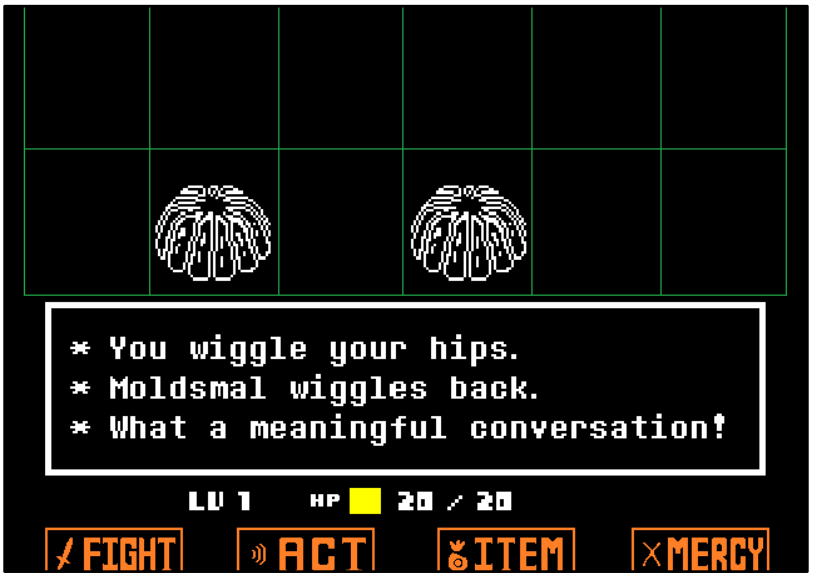

Figura 10: *Undertale.*

Fuente: https://undertale.com/about/.

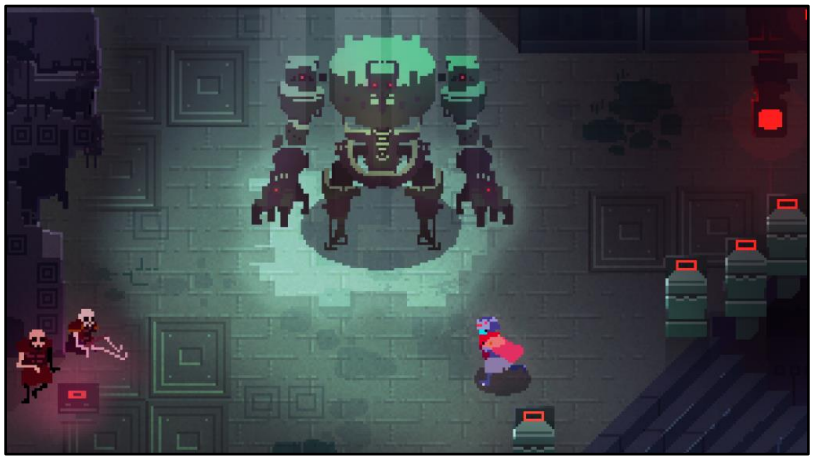

Figura 11: *Hyper Light Drifter.*
Fuente: https://www.orithegame.com.

- Unreal Engine [88]: Un motor de videojuegos multiplataforma, al igual que Unity, que fue mostrado por primera vez con el videojuego de disparos en primera persona, Unreal [89], en 1998, por la compañía Epic Games, conocida principalmente por la saga exclusiva de Microsoft, *Gears of War* [29], y últimamente, por el juego de moda, *Fortnite* [27]. Diferentes autores y desarrolladores (véase por ejemplo [56], [87] y [90]), han analizado las ventajas y desventajas entre utilizar uno u otro motor, considerándose ambos entre las mejores opciones para aprender a hacer videojuegos. Mientras que Unity ofrece a sus usuarios una amplia gama de herramientas y funciones a las que se puede acceder fácilmente, el catálogo de recursos de Unreal Engine 4 es menor y su aprendizaje más complejo. No obstante, la calidad gráfica de los videojuegos desarrollados con dicho motor es excepcional. De hecho, Unreal Engine 4 es actualmente uno de los motores más usados para videojuegos, tanto con un presupuesto muy alto, más conocidos como triple A, como para juegos realizados por desarrolladores independientes, también conocidos como indies. Un ejemplo de videojuego triple A podría ser el pendiente de salida a fecha de escritura de este documento, *Kingdom Hearts 3* [42] y cuya estética se muestra en la Figura 12 y un ejemplo de videojuego indie sería *A Hat in Time* [2], como se aprecia en la Figura 13.

  Este motor en su versión actual, la cual es la que se analizó como posible opción para realizar este proyecto, Unreal Engine 4, utiliza C++ como

lenguaje de programación, con el que no estoy aún del todo familiarizado, y no es un lenguaje sencillo. Por esta razón y teniendo en cuenta que este proyecto no está totalmente centrado en los videojuegos que se van a realizar, se descartó su uso, anticipándome así a los posibles problemas que pudieran generarse durante el desarrollo del proyecto y que previsiblemente la utilización de Unity no daría. Además de esto, Unreal Engine es gratuito hasta los 3.000 dólares de facturación. A partir de dicho precio, se deberá abonar el 5 por ciento del beneficio bruto.

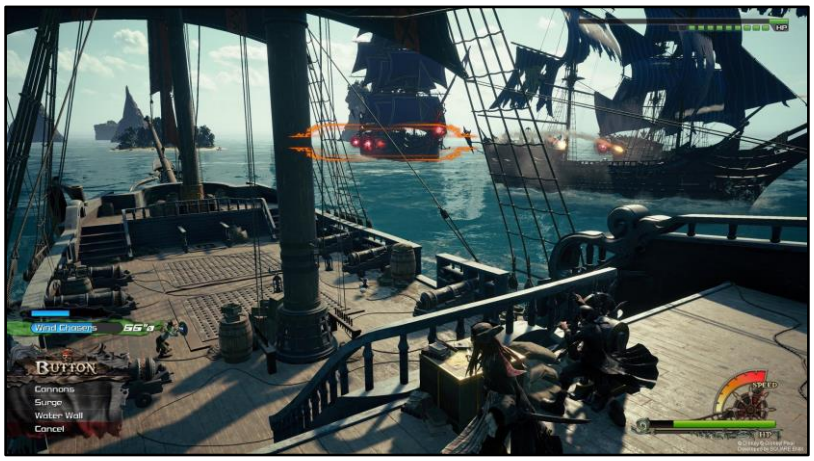

Figura 12: *Kingdom Hearts 3.*
Fuente: https://kingdomhearts.com/3/es/home/.

Figura 13: *A Hat in Time.*
Fuente: Captura del videojuego. http://hatintime.com/.

4.3.1.2. Herramientas de diseño 3D, Blender

Para el modelado de los diferentes elementos del videojuego *Penguin Rush* se utilizó Blender [12]. Blender permite diseñar objetos, personajes y escenas en 3D con diversas técnicas. La animación de los elementos se realiza mediante la técnica denominada keyframing o animación por fotogramas clave [79]. Esta herramienta de diseño se ha utilizado para realizar los modelos 3D que se usarán posteriormente como recursos en dicho videojuego. Se ha elegido Blender principalmente porque es gratuito y libre y, por lo tanto, no ha habido ningún gasto asociado a la obtención del mismo. Además, Blender fue el último programa de modelado 3D que se vio en el grado y, por lo tanto, me era más familiar que, por ejemplo, Autodesk 3DS Max [8] o Autodesk Maya [9]. Aun así, se analizaron dichas alternativas:

- Autodesk 3DS Max es una herramienta de modelado 3D general desarrollada por Autodesk. El problema principal por el que no fue elegido fue por su precio (aunque dispone de una versión gratuita para estudiantes), su tamaño y porque se ha dado prioridad a las herramientas de código abierto.
- Autodesk Maya es otra herramienta para modelado 3D realizada por Autodesk, pero está más enfocada a la animación.

En base a [48], donde se hace un estudio comparativo, tanto desde el punto de vista teórico como práctico, de dichas herramientas, se puede decir que la interfaz de Autodesk Maya es más intuitiva y que en relación al renderizado es la herramienta más potente de las tres. No obstante, en relación al factor texturas Blender supera a las demás.

4.3.1.3. Herramientas de desarrollo web, Node.js

La parte de servidor, más conocida como backend, es el corazón de la lógica de este proyecto. Esto significa que cualquier acción con la que interactuemos, como crear una variable, una métrica, añadir variables a las métricas o ver las métricas, por ejemplo, va a pasar por el servidor. Por tanto, es una de las partes más delicadas de este proyecto, y se debe tener cuidado al elegir la herramienta y lenguaje de programación que se va a utilizar. En este caso se ha decidido elegir Node.js. Dicho por la misma página web del proyecto [58], “Node.js®

es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome". Esto significa que es un sistema para poder programar aplicaciones de escritorio, o de línea de comandos utilizando JavaScript [40], lenguaje usado principalmente para web [35].

Este entorno fue elegido por la familiaridad con el mismo, así como con el lenguaje, JavaScript, debido a mi experiencia en prácticas de empresa. También se ha tenido en cuenta que es gratuito. Otras alternativas para este mismo proyecto podrían haber sido las siguientes:

- PHP (PHP: Hypertext Preprocessor) [64] es un lenguaje de código abierto que puede ser incrustado en HTML, y se puede usar como lenguaje de servidor con la ayuda de, por ejemplo, Apache [73]. Fue descartado por el hecho de que es un lenguaje que está yendo hacia el desuso, y en este proyecto se quería dar prioridad a nuevas tecnologías.
- ASP.NET [7] es una herramienta de desarrollo de aplicaciones web desarrollada por Microsoft. Es una herramienta sobre la cual se puede programar con varios lenguajes de programación. Se descartó por la poca familiaridad con dicha herramienta y los lenguajes permitidos.

Otra de las ventajas de Node.js es la inclusión de diferentes módulos que añaden funcionalidades. En este caso se han usado los siguientes módulos:

- CORS: framework para ayuda con las *Cross Origin Resource Sharing*, es decir, el poder realizar peticiones web desde diferentes sitios permitidos.
- Express: framework para aplicaciones web para manejar las peticiones que le llegan al servidor.
- Morgan: escribe en consola las peticiones que van llegando desde el frontend o desde los videojuegos.
- MySQL: este módulo es una librería de MySQL en JavaScript, para adaptar la persistencia a nuestro servidor. Se tratará con más detalle este módulo en la siguiente sección.

4.3.1.4. Herramientas de persistencia, MySQL

La persistencia de un sistema es, en pocas palabras, el hecho de que un sistema recuerde lo que ha realizado en ejecuciones pasadas, en nuestro caso particular, dónde vamos a guardar los datos de las métricas. Esto se consigue utilizando cualquier sistema que maneje datos y los almacene, siendo en este caso, una base de datos, y más concretamente MySQL.

MySQL [57] es un sistema de gestión de bases de datos relacional, basado en tablas. Ahora mismo es propiedad de Oracle, pero comenzó siendo de código abierto. Que sea un sistema relacional significa que la base de datos se podría considerar un conjunto de relaciones entre objetos, almacenados en tablas. Dichas tablas tienen atributos, o columnas, y objetos, o filas. Estos atributos pueden hacer referencia a otros objetos en otras tablas, y de ahí su potencia. Se puede observar esta estructura en el ejemplo de la Tabla 45 y la Tabla 46, de forma que las métricas 1, 2 y 3 pertenecen al juego 1, *Penguin Rush,* y la 4 al juego 2, *Pizza Clicker*.

Id_Juego	Nombre_Juego
1	Penguin Rush
2	Pizza Clicker

Tabla 45: *Ejemplo de tabla juego. Fuente: Elaboración propia.*

Id_Metrica	Id_Juego	Nombre_Metrica
1	1	Tiempo medio de vida
2	1	Sumatorio de muertes
3	1	Tiempo jugado
4	2	Pizzas hechas totales

Tabla 46: *Ejemplo de tabla métricas. Fuente: Elaboración propia.*

Como alternativa podríamos tener MongoDB [54], una base de datos que no es relacional sino basada en estructuras de objetos, usando una estructura similar a JSON, concretamente BSON (Binary JSON). En este sistema, en lugar de tener objetos relacionados entre sí, tenemos objetos que contienen otros objetos. Para clarificar, se podrá encontrar la estructura de objeto mostrada en la Figura 14, exactamente igual al ejemplo de las tablas de MySQL:

```
{
   "juegos": [
       {
           "nombre": "Penguin Rush",
           "metricas": [
               {
                   "nombre": "Tiempo medio de vida"
               },
               {
                   "nombre": "Sumatorio de muertes"
               },
               {
                   "nombre": "Tiempo jugado"
               }
           ]
       },
       {
           "nombre": "Pizza Clicker",
           "metricas": [
               {
                   "nombre": "Pizzas hechas totales"
               }
           ]
       }
   ]
}
```

Figura 14: *Objeto JSON.*

Fuente: Elaboración propia.

4.3.1.5. Herramientas de frontend, Angular y Bootstrap

La contraparte del backend es el frontend, es decir, la parte que ve, y con la que interacciona el usuario. En este proyecto la herramienta elegida para el frontend ha sido Angular 6 [4], debido a que simplifica las cosas sobremanera respecto a hacerlo en JavaScript puro. Por otra parte, para la apariencia y el diseño de la web se ha usado Bootstrap 4 [13], con el fin de simplificar la portabilidad del sistema a dispositivos móviles, de manera que sea una página web “responsive” o lo que es lo mismo adaptativa. Para las métricas se ha utilizado chart.js, herramienta a la que se puede acceder desde https://www.chartjs.org/.

Angular 6 es un framework para aplicaciones web, es decir, un conjunto de herramientas y funcionalidades que simplifican tareas repetitivas o complejas. En este caso, se ha utilizado principalmente para manejar la estructura de la página, además de la conexión con el servidor.

Se han descartado otras alternativas para la realización del frontend, como por ejemplo el uso de otros frameworks como React [67], Jquery [41] o Ruby [70], o incluso, sin ningún tipo de framework en absoluto, utilizando JavaScript puro. Esto se ha decidido así porque Angular es un framework sencillo pero potente.

4.3.1.6. Herramientas de control de versiones, GitHub

Una herramienta de control de versiones es prácticamente imprescindible para poder realizar proyectos de programación, ya que dichas herramientas son extremadamente útiles para llevar un seguimiento del código, tanto actual como pasado y permite realizar acciones como revertir el código a un estado anterior o compartir el código con otros programadores o dispositivos.

En este proyecto para realizar dicho control de versiones se ha utilizado Git, y en particular el servicio que proporciona GitHub [30], ya que la creación y mantenimiento de los repositorios es gratis para estudiantes. Para el resto, el servicio es totalmente gratuito si sus proyectos son abiertos, es decir, que todo el mundo puede ver el código.

Ahora mismo en el mercado podríamos encontrar alternativas a GitHub que podría haber elegido sin problemas, debido a que estas son gratuitas. Estas alternativas son:

- GitLab [31], que nos permite tener repositorios gratuitos y privados para siempre, además de la posibilidad de poder montar nuestro propio servidor de Git.
- Bitbucket [11], que proporciona repositorios de código gratuitos, pero solo durante un tiempo. Pasado este, los repositorios se bloquean y no se puede acceder a ellos hasta que se pague una suscripción.

4.3.2. Herramientas hardware

Adicionalmente a las herramientas software tratadas en la sección anterior, para el desarrollo de este proyecto, se han usado diferentes herramientas hardware; entre ellas se encuentran un PC de sobremesa de gama media-alta y un portátil con prestaciones muy bajas, pero pequeño y con batería de alta duración.

Respecto a accesorios, se han usado los típicos en cualquier computador adaptado para trabajar, quizá destaquen el uso de múltiples monitores y teclado mecánico. Además de esto se han usado, para algunos assets, una tableta digitalizadora.

4.4. Ámbito de uso y público objetivo

El público objetivo de este sistema es el de desarrolladores de videojuegos que quieren tener más información sobre cómo exprimir más sus videojuegos, tanto a nivel de diversión como de ventas de micropagos.

Este sistema se utilizaría únicamente en el ámbito de desarrollo de videojuegos en principio, pero se podría extrapolar a cualquier otro sistema que se pueda adaptar al envío de peticiones HTTP como, por ejemplo, un sistema de calificación de servicio como podemos encontrar últimamente en tiendas, aeropuertos, centros comerciales, etc.

4.5. Metodología, el método Scrum

Una metodología es un conjunto de normas y procedimientos racionales para alcanzar un objetivo en cualquier ámbito. La elección de una metodología de trabajo adecuada para cualquier proyecto puede significar la diferencia entre el éxito o el fracaso.

La metodología por la que nos hemos decantado en este proyecto está basada en una de las metodologías ágiles, es decir, basada en el desarrollo iterativo o incremental y concretamente en una de las metodologías más usadas en el

entorno del desarrollo de sistemas informáticos. Concretamente dicha metodología es Scrum.

Esta metodología está basada en un conjunto de métodos y roles definidos. Concretamente, el término sprint se utiliza para definir las iteraciones de la metodología Scrum. Cuando un sprint termina, otro sprint comienza. Al final de cada sprint, se debe obtener un resultado completo, es decir, una versión de una parte funcional del proyecto o mini proyecto. De esta forma se van incrementando las características del producto en cada sprint. Una versión del producto final se puede conseguir al final de un sprint o después de varios sprints.

Respecto a los roles se tiene el “Product Owner” o cliente, es decir, para quién se hace el producto, y que será fuente de requisitos, el equipo, quién hará el sistema, y el “ScrumMaster” o facilitador que hará de puente entre ambos, además de procurar que se sigan las reglas de Scrum y gestionar posibles cambios en el proyecto.

Teniendo en cuenta que este proyecto es un trabajo fin de grado que además fue propuesto por el propio autor a la tutora, se ha utilizado para su realización una variación de la metodología Scrum en la que he asumido todos los roles del equipo de desarrollo bajo la supervisión de la tutora, que en ocasiones ha hecho el papel de “ScrumMaster”.

En particular, el haber aplicado Scrum a este proyecto significa que se han realizado iteraciones sobre el mismo producto, ya sea videojuegos, servidor, o página web en sprints en los que al final se ha tenido una versión cada vez con más funcionalidades que se han definido en los requerimientos y en los casos de uso.

4.6. Planificación temporal

Al haber utilizado una metodología ágil para desarrollar este proyecto no es posible realizar una planificación temporal exhaustiva. Aun así, en base a los objetivos, tanto generales como específicos, que se pretendían alcanzar con la realización de este proyecto se definieron las tareas a realizar y el tiempo estimado para su realización.

En la Figura 15 y en la Tabla 47 se explica esta planificación. Concretamente, en la Tabla 47 se resumen las tareas propuestas para el desarrollo del proyecto y en la Figura 15 la dedicación temporal estimada mediante la ayuda de un diagrama de Gantt.

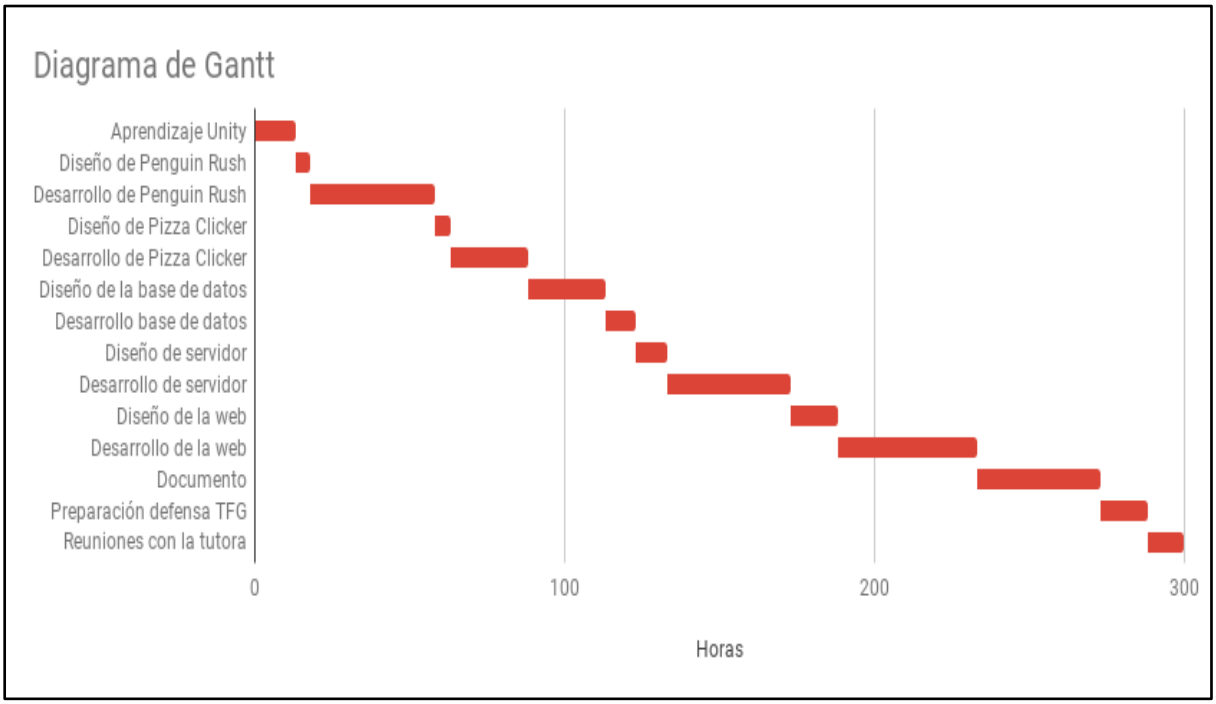

Figura 15: *Diagrama de Gantt.*

Fuente: Elaboración propia.

Tarea	Fecha de inicio	Fecha de fin	Tiempo estimado
Aprendizaje de Unity	03/01/2018	22/01/2018	13 horas
Diseño de Penguin Rush	03/01/2018	05/01/2018	5 horas
Desarrollo de Penguin Rush	08/01/2018	15/02/2018	40 horas
Diseño de Pizza Clicker	04/06/2018	06/06/2018	5 horas
Desarrollo de Pizza Clicker	07/06/2018	30/06/2018	25 horas
Diseño de la base de datos	01/07/2018	10/07/2018	25 horas
Desarrollo de la base de datos	11/07/2018	15/07/2018	10 horas
Diseño del servidor	16/07/2018	22/07/2018	10 horas
Desarrollo del servidor	23/07/2018	15/08/2018	40 horas
Diseño de la página web	16/08/2018	31/08/2018	15 horas
Desarrollo de la página web	01/09/2018	16/09/2018	45 horas
Documento	17/09/2018	18/11/2018	40 horas
Preparación defensa TFG	19/11/2018	11/12/2018	15 horas
Reuniones con tutora	19/12/2017	11/12/2018	12 horas

Tabla 47: *Planificación de tareas.*

Fuente: Elaboración propia.

5. Diseño y desarrollo

En este capítulo se describirán los diferentes desarrollos que se han realizado para este proyecto, desde los videojuegos hasta el sistema de gestión de métricas de la web.

5.1. Videojuegos

En esta sección se exponen los pasos que se han realizado para el desarrollo de los dos videojuegos, *Penguin Rush y Pizza Clicker*, tanto aquellos que son comunes a ambos videojuegos como los específicos de cada videojuego en particular.

5.1.1. Concepto y bocetos

Cuando se empezó a realizar este proyecto, uno de los primeros pasos comunes a sendos desarrollos de los videojuegos fue decidir la mecánica principal y la estética general.

Para la mecánica principal de *Penguin Rush* se decidió hacer un videojuego en el que un personaje cayese continuamente por un tobogán y que la mecánica principal fuera esquivar obstáculos y evitar caer del tobogán.

Respecto a *Pizza Clicker*, la idea original era hacer un clicker de pizzas, donde se pudiese elegir qué ingredientes llevará la pizza y hacer cientos de ellas, venderlas con un simple botón y ganar recursos con los que comprar mejoras, que a su vez aumentarán la cantidad de recursos que obtendremos.

Para el diseño preliminar se realizaron diferentes bocetos, algunos de los cuales pueden observarse en las figuras siguientes. Dichos bocetos son en su totalidad para *Penguin Rush*, ya que para *Pizza Clicker*, al beber tanto de *Cookie Clicker* [18] no fueron necesarios.

Concretamente, en la Figura 16 se puede observar el primer boceto del pingüino y su adaptabilidad a un estilo más cuadriculado de personaje. Este diseño no me convencía, pero sentaría las bases para el siguiente concepto, que se puede ver en la Figura 17. En este ya observamos lo que sería el concepto final de personaje, atractivo y agradable a la vista.

Respecto al diseño de niveles, se realizó durante la programación del videojuego, sin prácticamente bocetos en papel, pero basándome en las formas que quería en la pantalla, como se puede ver en el boceto de la Figura 18.

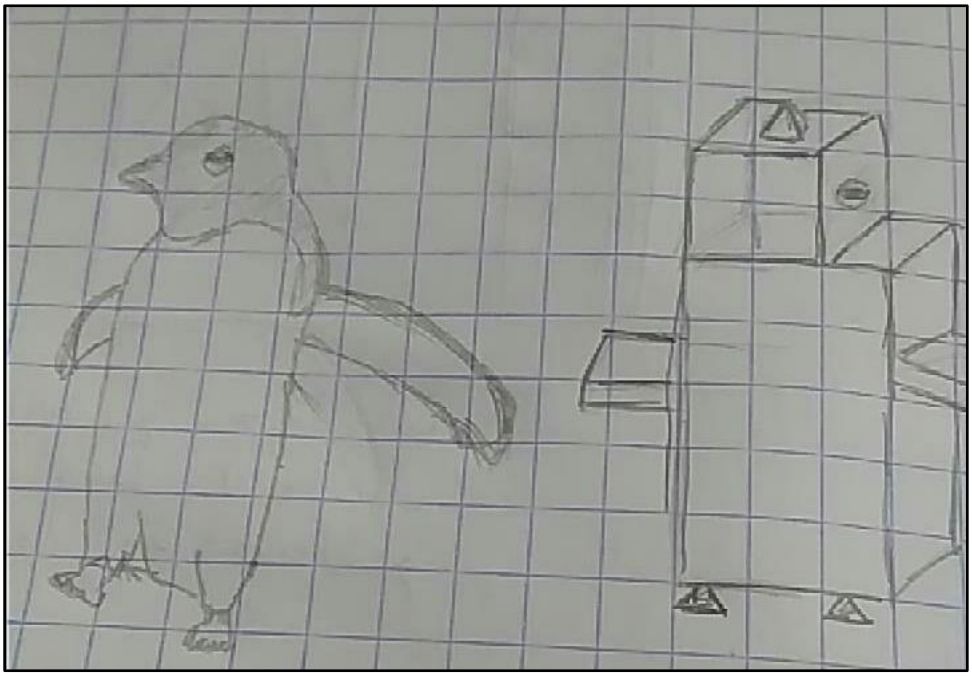

Figura 16: *Bocetos iniciales de personaje en Penguin Rush.*
Fuente: Elaboración propia.

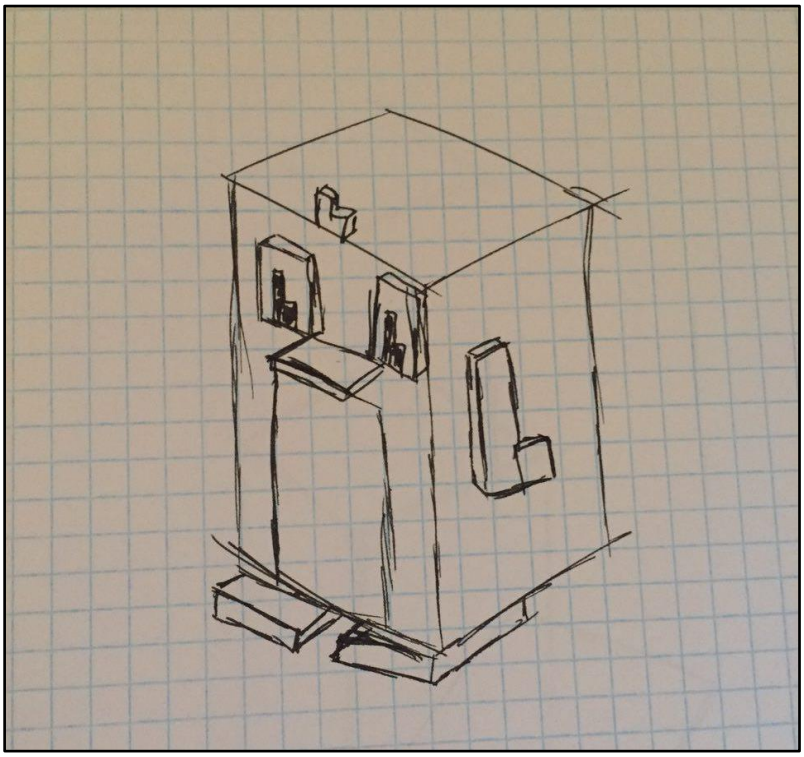

Figura 17: *Boceto final de personaje en Penguin Rush.*

Fuente: Elaboración propia.

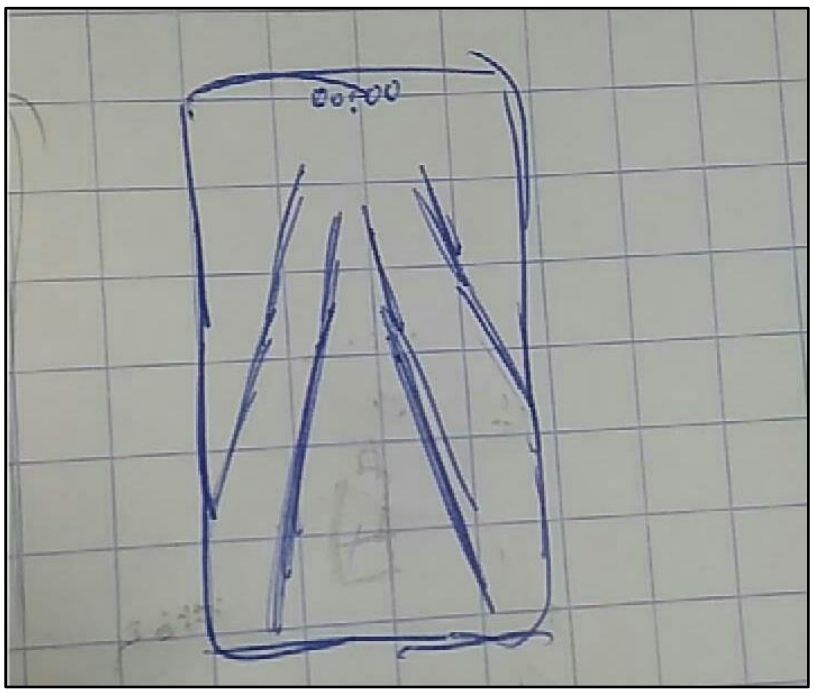

Figura 18: *Boceto final de diseño de nivel de Penguin Rush.*

Fuente: Elaboración propia.

5.1.2. Inicio de desarrollo

Para cualquier videojuego en Unity, el primer paso es crear el proyecto. Para ello hay que descargarse el motor desde la página oficial, y al instalarlo y abrirlo encontramos una interfaz, donde se puede abrir, crear o borrar un proyecto. En este caso se va a crear, y se verá algo como lo que aparece en la Figura 19, donde habrá que darle nombre al proyecto, decidir qué plantilla se usará, si 2D o 3D, y decidir otros factores, como qué paquetes se van a tener en el proyecto, o dónde va a estar localizado.

Una vez realizado este paso, solo es necesario abrir Unity y elegir el nuevo proyecto.

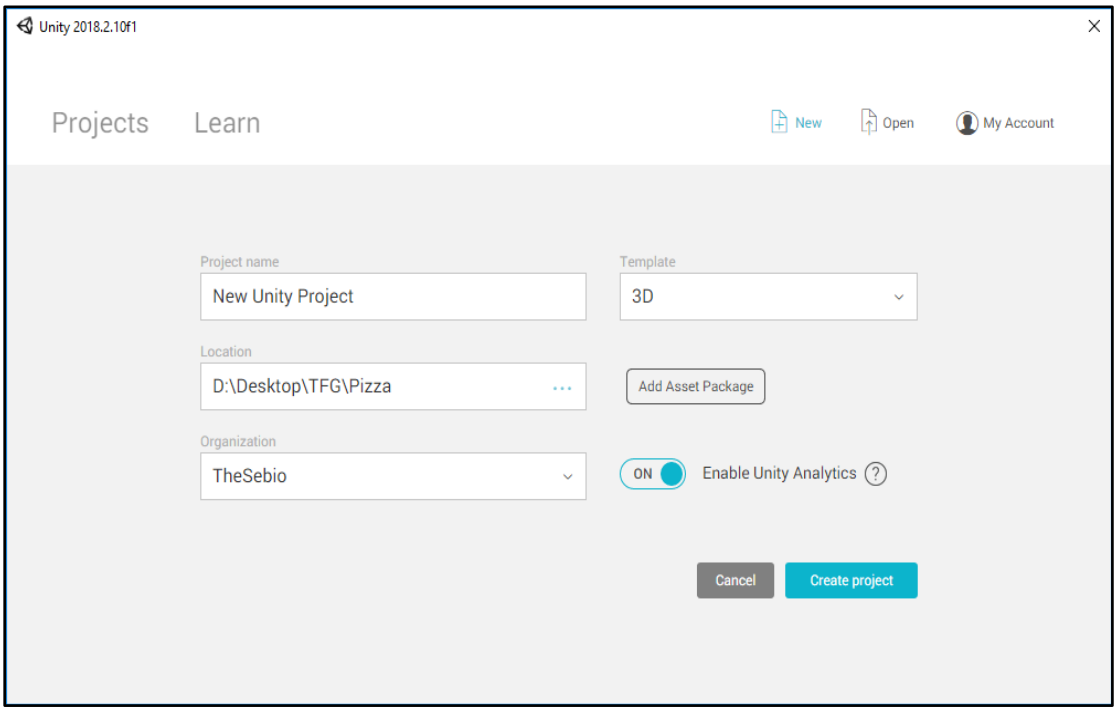

Figura 19: *Interfaz de creación de Unity.*

Fuente: Elaboración propia.

5.1.3. Desarrollo de Penguin Rush

En esta sección se explica el desarrollo llevado a cabo para *Penguin Rush* (véase Figura 20), así como los modelados usados en el mismo.

Figura 20: *Juego Penguin Rush.*
Fuente: Elaboración propia.

5.1.3.1. Clases y Scripts en Penguin Rush

Para el desarrollo de *Penguin Rush*, lo primero que se debe hacer es crear una clase que se encargue de manejar la creación de terreno infinito, en este caso, toboganes. Para ello se define la clase TileManager cuyo código se incluye en la Figura 21. Como se puede observar, tenemos diferentes atributos en esta clase, pero ninguna funcionalidad, por lo que esta clase no hace nada ahora mismo. Por tanto, se usarán los eventos de Unity para darles funcionalidad.

```
public class TileManager : MonoBehaviour {

   public GameObject[] tilePrefabs; //aquí guardaremos los diferentes
   diseños de tobogán que tendremos en Unity

   private Transform playerTransform; //donde está el jugador
   private float spawnZ = -25.0f; //donde aparecen los bloques en Z
   private float spawnY = 0.0f; //donde aparecen los bloques en Y

   public float tileLenght = 12.0f;//longitud de los bloques
   public float tileHeight = 12.0f;//altura de los bloques

   public int tilemax = 14; //máximo de bloque en pantalla
   private List<GameObject> activeTiles; //lista para guardar los bloque
   public float safeZone = 60; //zona que se necesita sobrepasar para borrar
un bloque
   private int rep = -1, last = -1; //para evitar repetirnos
```

Figura 21: *Definición de la clase TileManager.*

Fuente: Elaboración propia.

El primer evento que veremos será Start (véase Figura 22), evento que se ejecutará al principio de la escena. En esta función, lo primero que se hace es buscar dónde está el jugador y decirle al sistema que coja su posición para tenerla en cuenta a la hora de generar las construcciones. Una vez hecho esto, se inicializan los bloques y se crean con la función SpawnTile (función que se tratará más tarde), pero sin dejar los seis primeros bloques al azar de forma que aparezcan los bloques básicos (bajada simple). Esto se realiza de esta forma para que el usuario pueda habituarse al juego.

El evento Update (véase Figura 23) se llama cada vez que se renderiza cada fotograma.

En esta función, para dicho evento, tenemos la lógica que hace el juego infinito, es decir, una creación constante de bloques si el usuario ha avanzado lo suficiente en la “zona segura”. Además, una vez introducido un bloque nuevo, con la ayuda de la función DeleteTile, se borra el primero de la lista, que, gracias al atributo de zona segura, el jugador lo pasó hace rato.

```
    void Start () {

        playerTransform =
gameObject.FindGameObjectWithTag("Player").transform; //asignamos a
playerTransform donde está el jugador
        activeTiles = new List<GameObject>(); //inicializamos activeTiles
        for (int i = 0; i < tilemax; i++) //y ahora generamos los toboganes
        {
            if (i < 6)  //los primeros 6 bloques los hacemos planos
            {
                SpawnTile(5);
            }
            else //y el resto aleatorios
            {
                SpawnTile();
            }
        }
    }
```

Figura 22: *Definición del evento Start de la clase TileManager.*

Fuente: Elaboración propia.

```
    void Update () {
        if (GameObject.FindGameObjectWithTag("Player")) //si el jugador
existe
        {
            //comprobamos que tenemos que meter un nuevo bloque
            if (playerTransform.position.z - safeZone > (spawnZ - tilemax * t
ileLenght))
            {
                SpawnTile(); //metemos el bloque
                DeleteTile(); //y borramos un bloque
            }
        }
    }
```

Figura 23: *Definición del evento Update de la clase TileManager.*

Fuente: Elaboración propia.

La función SpawnTile (véase Figura 24) crea un nuevo bloque delante del resto de bloques, apoyándose en un bloque auxiliar, que se va editando para que vaya en su sitio, y finalmente se coloca en la lista de bloques activos para que se muestre.

El resto de funciones del generador son demasiado sencillas como para exponer el código, como por ejemplo, deleteTile, que simplemente destruye el primer bloque de la lista, o getSpawnY y getSpawnZ, que nos dan la posición en la que se deberán colocar los siguientes bloques, en el eje Y (componente horizontal) y el Z (profundidad), respectivamente.

```
   void SpawnTile(int prefabIndex = -1)
   {
       GameObject go; //esto es un objeto temporal donde almacenaremos el
bloque
       if (prefabIndex == -1) //si no nos han dado un bloque por parametros
entramos aquí
       {
           prefabIndex = Random.Range(0, tilePrefabs.Length);
//aleatorizamos el bloque
       }
        //creamos el bloque a partir del array de bloques
       go = Instantiate(tilePrefabs[prefabIndex]) as GameObject;
       //obtenemos la altura y el largo del bloque
       tileLength = go.GetComponent<TileData>().GetTileWidth();
       tileHeight = go.GetComponent<TileData>().GetTileHeight();
       //y lo colocamos donde nos convenga
       go.transform.position = Vector3.forward * (tileLength +spawnZ) +
Vector3.down * (tileHeight + spawnY);
       //y aumentamos la distancia donde se tiene que crear el siguiente
bloque
       spawnY += tileHeight;
       spawnZ += tileLength;
       //y finalmente, añadimos go
       activeTiles.Add(go);
   }
```

Figura 24: *Definición del método SpawnTile de la clase TileManager.*

Fuente: Elaboración propia.

El siguiente paso es la creación de una clase jugador (PenguinRuning), en la que se almacena el control del personaje. Dicha clase se define en la Figura 25.

```
public class PenguinRuning : MonoBehaviour {
   public Camera cam;
   public float startSpeed;
   public float frontPushForce;
   public float sidePushForce;

   public float seconds;
   public float lastTouchTime;
   public float touchsPerSecond;
```

Figura 25: *Definición de la clase PenguinRuning.*

Fuente: Elaboración propia.

En esta clase se almacena la cámara (que se tratará posteriormente), la velocidad de inicio, la velocidad que se le dará cuando se toca la pantalla, tanto hacia delante con frontPushForce como para los lados con sidePushForce. Por último se definen tres variables cuyo uso queda relegado a limitar la cantidad de empujes que puede haber por segundo, ya que en caso contrario, un toque durante un segundo equivaldría a 60 toques debido a la tasa de fotogramas por segundo y su ejecución.

En esta clase, el método Start define el inicio del personaje, dándole una velocidad inicial y pasando de toques que se pueden realizar por segundo a cada cuánto se puede realizar un toque, por lo que se debe dividir 1 entre la cantidad de toques por segundo (véase Figura 26).

```
   void Start () {
       GetComponent<Rigidbody>().AddForce(0, 0, frontPushForce*5,
ForceMode.Impulse);
       touchsPerSecond = 1 / touchsPerSecond;
       lastTouchTime = 0;
   }
```

Figura 26: *Definición del evento Start de la clase PenguinRuning.*

Fuente: Elaboración propia.

En Update, que, recordemos, se ejecuta cada iteración del videojuego, se calculan los segundos de juego, además de llamar a la función control (véase Figura 27).

```
void Update () {
    control();
    seconds = seconds + Time.deltaTime * 10;
}
```

Figura 27: *Definición del evento Update de la clase PenguinRuning.*

Fuente: Elaboración propia.

Dicha función se encarga de realizar todas las acciones necesarias para que el control funcione correctamente (véase Figura 28).

```
void control()
{
    bool mytouch = Input.GetMouseButton(0);
    Vector2 pos = Input.mousePosition;
    Vector3 pinguPosScreen = cam.WorldToScreenPoint(transform.position);
    if (lastTouchTime + touchsPerSecond >= seconds)
    {
        mytouch = false;
    }
    if (mytouch == true)
    {
        lastTouchTime = seconds;
        PinguPush(pos.x, pinguPosScreen.x);
    }
}
```

Figura 28: *Definición del método control de la clase PenguinRuning.*

Fuente: Elaboración propia.

En esta función de la clase jugador se maneja el control del mismo mediante toques en la pantalla, controlando que no haya más toques por segundo de los definidos. Una vez se efectúa un toque en la pantalla llamamos a PinguPush, encargado de ver si se mueve el pingüino a la derecha o hacia la izquierda, dependiendo de dónde se toque y la posición del pingüino. Dicho método se puede ver en el trozo de código expresado en la Figura 29.

```
    private void PinguPush(float touch, float pingu)
    {
        if(touch > pingu)
        {
            GetComponent<Rigidbody>().AddForce(-sidePushForce, 0,
frontPushForce, ForceMode.Impulse);
        }
        else
        {
            GetComponent<Rigidbody>().AddForce(sidePushForce, 0,
frontPushForce, ForceMode.Impulse);
        }
    }
```

Figura 29: *Definición del método PingPush de la clase PenguinRuning.*

Fuente: Elaboración propia.

La siguiente y última clase de *Penguin Rush* es CameraScript, dedicada a controlar la cámara para que siempre siga al jugador, además de contabilizar los puntos que acumula el jugador y controlar cuando pierde (véase Figura 30).

```
public class CameraScript : MonoBehaviour {
    public GameObject player;       //variable pública para almacenar el
jugador
    public GameObject tileManager;       //variable pública para almacenar el
terreno


    private Vector3 offset;         //distancia entre jugador y cámara
    private float initialZ;

    private float points;           //puntos
    private float oldPoints;

    public GameObject PointsText;
    public GameObject FinalPoints;

    public GameObject CanvasUIOnGame;
    public GameObject CanvasUIOnLose;

    public GameObject retryButton;
```

Figura 30: *Definición de la clase CameraScript.*

Fuente: Elaboración propia.

Se definen como atributos de esta clase los siguientes:

- **Player**: el jugador, para poder acceder a atributos de su clase como, por ejemplo, su posición.
- **TileManager**: el terreno, para controlar cuando pierde el jugador por caída hacia abajo.
- **Offset**: distancia entre jugador y cámara, si los valores de este vector son pequeños, la cámara estará muy pegada al mismo.
- **Initial Z**: distancia desde donde empieza el jugador para tener en cuenta los puntos.
- **Points**: puntos que lleva el jugador ahora mismo.
- **OldPoints**: variable de control cuya finalidad es mantener el máximo de puntos si el jugador los pierde (se da la vuelta momentáneamente).
- **PointsText**: objeto de interfaz que muestra la cantidad de puntos actuales al jugador.
- **CanvasUIOnGame**: canvas donde se almacena la interfaz gráfica que se muestra mientras se está jugando.
- **CanvasUIOnLose**: canvas donde se almacena la interfaz gráfica que aparece cuando se pierde.
- **RetryButton**: botón de reinicio del videojuego. Solo aparece cuando se ha perdido.

El evento de inicio de esta clase está compuesto por buscar el jugador y guardarlo en la clase, calcular el offset con la resta de la distancia de la cámara y el jugador, y definir la Z inicial (profundidad inicial) y el texto de los puntos (véase Figura 31).

En el evento Update, definido en la Figura 32, se maneja el movimiento de la cámara, con el objetivo de que esté siempre siguiendo al jugador en el eje Z, es decir, en la profundidad, y en el eje Y, es decir, en la componente horizontal.

```
    void Start () {
        player = GameObject.FindGameObjectWithTag("Player");
        offset = transform.position - player.transform.position;
        initialZ = transform.position.z;
        PointsText = GameObject.Find("Text_Points");
    }
```

Figura 31: *Definición del evento Start de la clase CameraScript.*

Fuente: Elaboración propia.

```
    void Update () {
        if (player)
        {
            Vector3 newPosition = transform.position;
            newPosition.z = player.transform.position.z + offset.z;
            newPosition.y = player.transform.position.y + offset.y;
            transform.position = newPosition;
            points = (newPosition.z - initialZ)*10;
            if (points >= oldPoints)
            {
                PointsText.GetComponent<Text>().text = points.ToString("F0");
                oldPoints = points;
            }
            else
            {
                PointsText.GetComponent<Text>().text = points.ToString("F0");
                points = oldPoints;
            }
            float spawnY =
tileManager.GetComponent<TileManager>().getSpawnY();
            if (-player.transform.position.y > spawnY)
            {
                lose(points);
            }
        }
    }
```

Figura 32: *Definición del evento Update de la clase CameraScript.*

Fuente: Elaboración propia.

Además de esto, se manejan los puntos, siendo estos la posición actual menos la posición origen por diez para quitar decimales. Después de esto, se controla que los puntos se queden en el máximo, aunque se ande hacia atrás, y también la condición de derrota, si el jugador está, en algún momento, en una posición por debajo de la posición en el eje Y donde aparecerá el bloque. Es entonces cuando se llama a la función lose, que se define en la Figura 33.

```
void lose(float p)
{
    Destroy(player);
    CanvasUIOnLose.SetActive(true);
    FinalPoints = GameObject.Find("Text_Points_Final");
    FinalPoints.GetComponent<Text>().text = p.ToString("F0");
    Button btn = retryButton.GetComponent<Button>();
    btn.onClick.AddListener(retryButtonOnClick);
}
```

Figura 33: *Definición del método Lose.*

Fuente: Elaboración propia.

La primera acción que se realiza cuando se pierde es borrar el jugador, para que deje de caer y liberar memoria. Una vez borrado, se activa la interfaz de derrota y se calculan los puntos finales. Por último, se prepara un evento para cuando se pueda pulsar el botón, que volverá a lanzar el juego mediante la función retryButtonOnClick (véase Figura 34). Esta función busca como se llama la escena actual y la carga de nuevo, empezando así el juego desde el principio.

```
void retryButtonOnClick()
{
    Scene scene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(scene.name);
}
```

Figura 34: *Definición del método retryButtonOnClick.*

Fuente: Elaboración propia.

5.1.3.2. Modelados para Penguin Rush

En esta sección se incluyen los renders de los modelados de *Penguin Rush* realizados en Blender. Concretamente la Figura 35 muestra el personaje jugador de *Penguin Rush,* mientras que el resto (véase Figura 36-Figura 40) están dedicadas a los distintos tipos de bloques que aparecen en el juego.

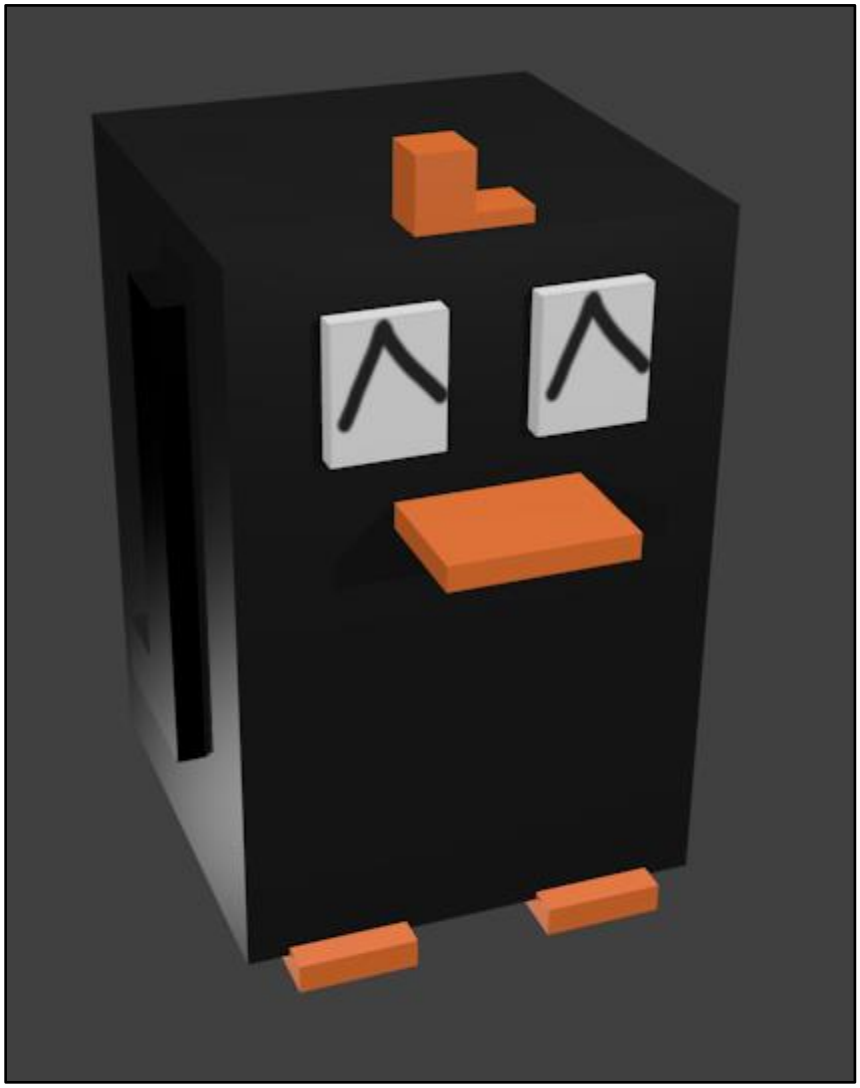

Figura 35: *Personaje jugador de Penguin Rush.*
Fuente: Elaboración propia.

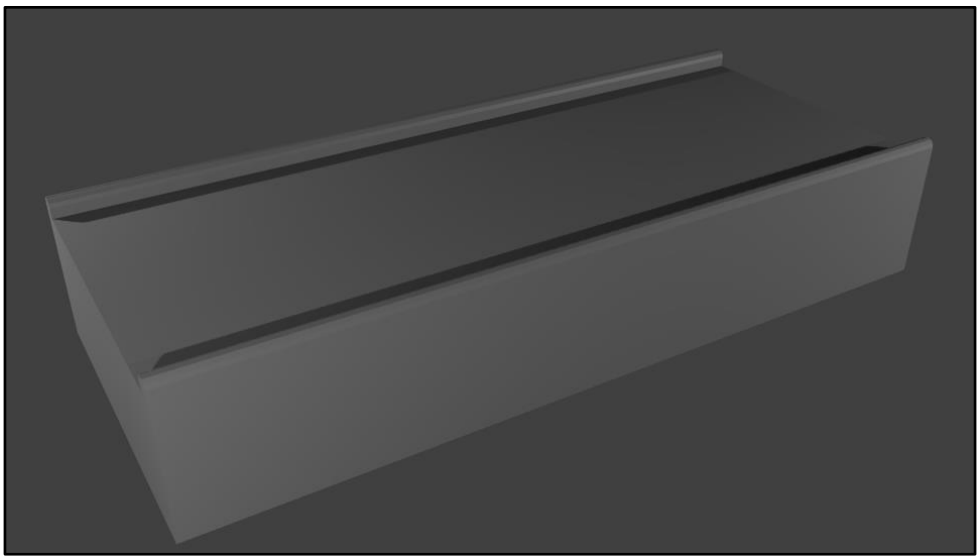

Figura 36: *Bloque 1 Penguin Rush.*

Fuente: Elaboración propia.

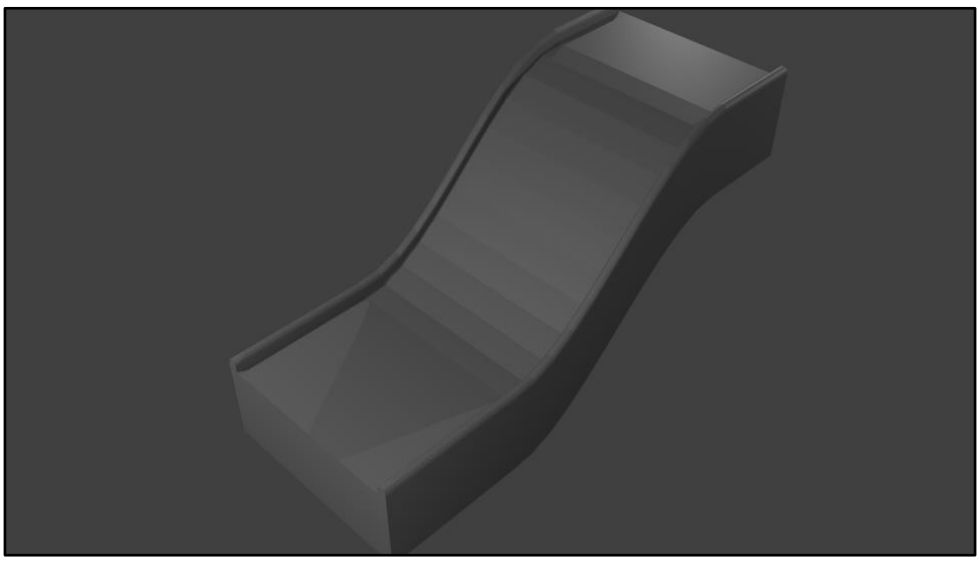

Figura 37: *Bloque 2 Penguin Rush.*

Fuente: Elaboración propia.

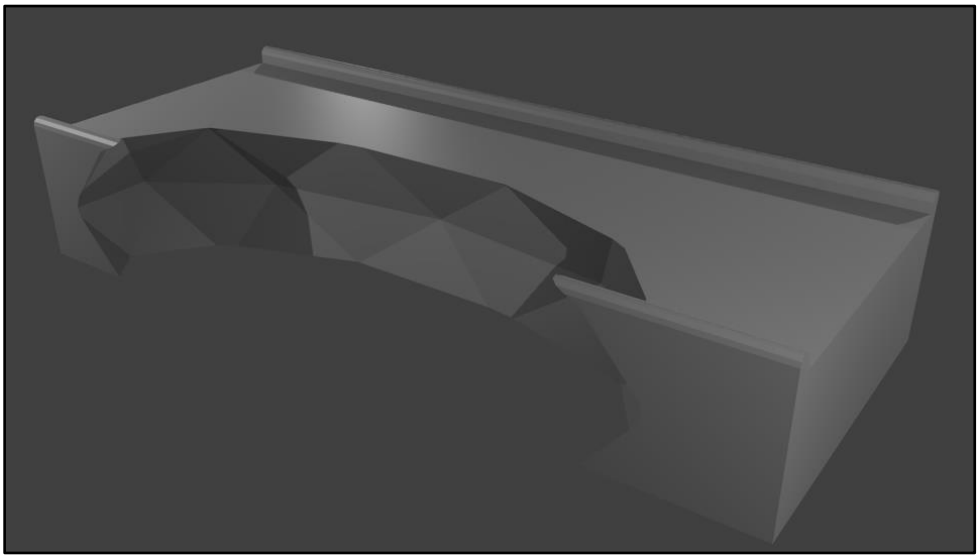

Figura 38: *Bloque 3 Penguin Rush.*

Fuente: Elaboración propia.

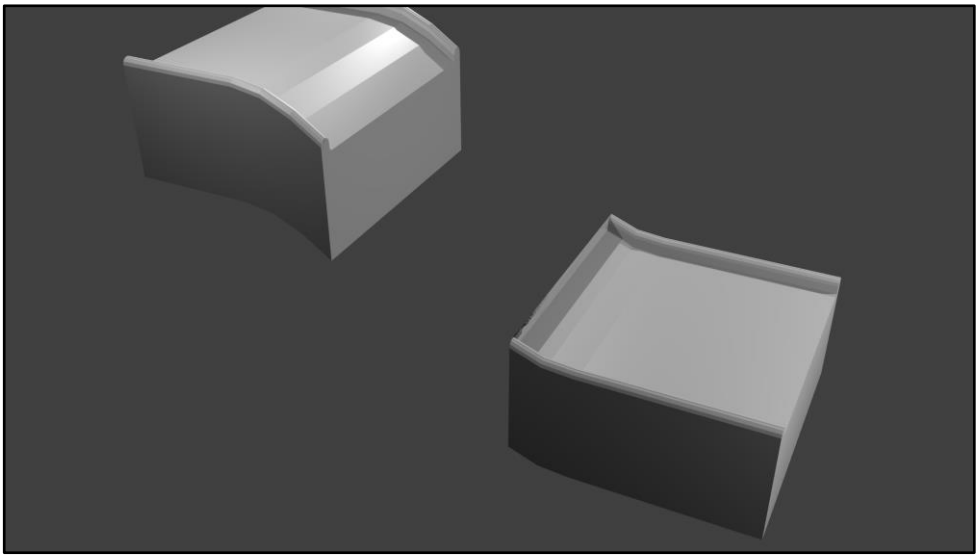

Figura 39: *Bloque 4 Penguin Rush.*

Fuente: Elaboración propia.

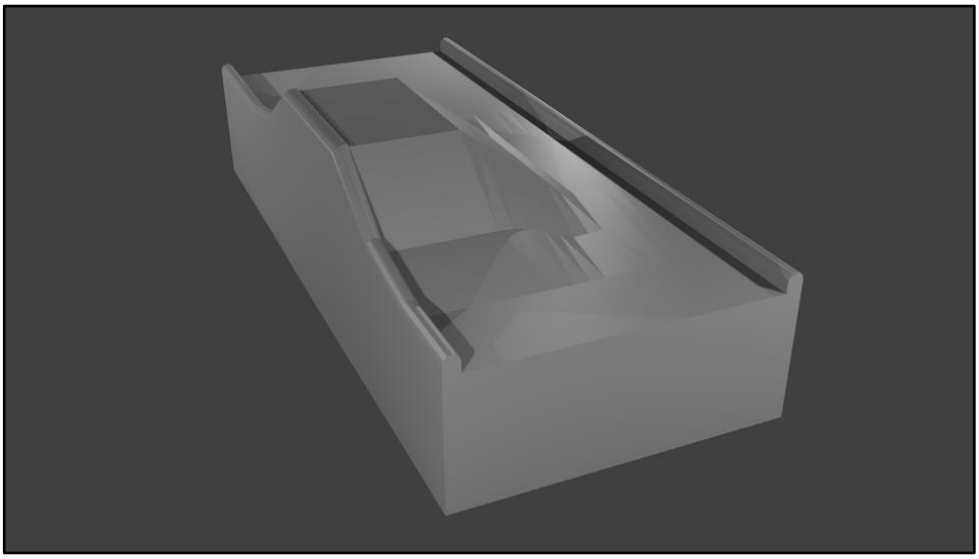

Figura 40: *Bloque 5 Penguin Rush.*

Fuente: Elaboración propia.

5.1.4. Desarrollo de Pizza Clicker

En esta sección se explica el desarrollo de *Pizza Clicker (véase* Figura 41).

Figura 41: *Juego Pizza Clicker.*

Fuente: Elaboración propia.

Se sigue una estructura similar a *Penguin Rush*, ya que ambos juegos fueron realizados con el mismo motor, como se ha mencionado anteriormente.

5.1.4.1. Clases y Scripts en Pizza Clicker

En este juego hay tantas clases como ingredientes se pueden añadir a una pizza, por lo que, solo se expondrá una de las clases, la mostrada en la Figura 42.

En esta clase se define el componente de interfaz que se usa, en thisDrop, además de definir los tipos de masa que se podrán elegir. Por último se define una variable que controla cuántas veces se ha cambiado la masa.

En el método Start() se inician los objetos masaNormal, masaFina y masaGruesa a false, por lo que al principio no habrá ninguna masa. Esto se podrá cambiar posteriormente con la función CreateMasaSprite, poniendo la masa que seleccionemos a true. El resto de clases de ingredientes son muy parecidos a la clase MasaSpriteScript y por tanto no se han incluido en el documento.

Respecto a la clase Click (véase Figura 43), que utilizaremos para manejar la pulsación del botón, se definen los diferentes elementos que se mostrarán, como los ingredientes, o los textos, además de la cantidad de pizzas que se tiene, el dinero que se añadirá por pizzas y las pizzas que se crearán al pulsar el botón.

```
public class MasaSpriteScript : MonoBehaviour {
    public UnityEngine.UI.Dropdown thisDrop;
    public UnityEngine.GameObject masaNormal;
    public UnityEngine.GameObject masaFina;
    public UnityEngine.GameObject masaGruesa;
    public static int masaSpriteClickedCount = 0;

    public void Start()
    {
        masaNormal.SetActive(false);
        masaFina.SetActive(false);
        masaGruesa.SetActive(false);
    }

    //Masa sprites
    public void CreateMasaSprite()
    {
        if (masaSpriteClickedCount < 1)
        {
            if (thisDrop.value == 1)
            {
                masaNormal.SetActive(true);
            }

            if (thisDrop.value == 2)
            {
                masaFina.SetActive(true);
            }

            if (thisDrop.value == 3)
            {
                masaGruesa.SetActive(true);
            }
            masaSpriteClickedCount++;
        }
    }
}
```

Figura 42: *Definición de clase MasaSpriteScript.*

Fuente: Elaboración propia.

```
public class Click : MonoBehaviour {

   public UnityEngine.UI.Text pizzaDisplay;
   public UnityEngine.UI.Text moneyDisplay;
   public UnityEngine.UI.Dropdown masaDrop;
   public UnityEngine.UI.Dropdown quesoDrop;
   public UnityEngine.UI.Dropdown salsaDrop;
   public UnityEngine.UI.Dropdown toppingDrop;
   public float pizzas = 0.00f;
   public static int pizzaPerClick = 1;
   private float addedMoney = 0.00f;
   public void Update ()
   {
       pizzaDisplay.text = "Pizzas sold: " + pizzas;
   }
   public void Clicked() {
       if (PizzaAssembly.assemblyList.Count >= 2)
       {
           // Cada vez que vendemos una pizza reseteamos el dinero que vamos
a añadir
           addedMoney = 0.00f;
           // Mediante assemblyText obtenemos los nombres de los
ingredientes
           GameObject theMasaButton = GameObject.Find("Masa Dropdown");
           PizzaAssembly assemblyText =
theMasaButton.GetComponent<PizzaAssembly>();
           pizzas += pizzaPerClick;
           // Añadir el dinero de la pizza
           for (int i = 0; i < MasaMoneySystem.list2.Count; i++)
           {
               addedMoney = addedMoney + MasaMoneySystem.list2[i];
           }
           MasaMoneySystem.money = MasaMoneySystem.money + addedMoney;
           moneyDisplay.text = "Money: " +
MasaMoneySystem.money.ToString("f2");
       }
   }
}
```

Figura 43: *Definición de clase Click.*

Fuente: Elaboración propia.

El método Clicked de esta clase se llamará cuando se pulse el botón de vender pizza. En dicho método se realizan las siguientes acciones en orden:

- Se restablece addedMoney, es decir el dinero que se obtiene por una pizza.
- Se busca cada ingrediente y se guarda el dinero de los ingredientes en addedMoney.
- Se añade el dinero de la pizza y la cantidad de pizzas hechas.

5.1.4.2. Sprites para Pizza Clicker

A continuación se muestran algunos sprites correspondientes a salsas para las pizzas (véase Figura 44), a toppings (véase Figura 45) y a tipos de masa (véase Figura 46).

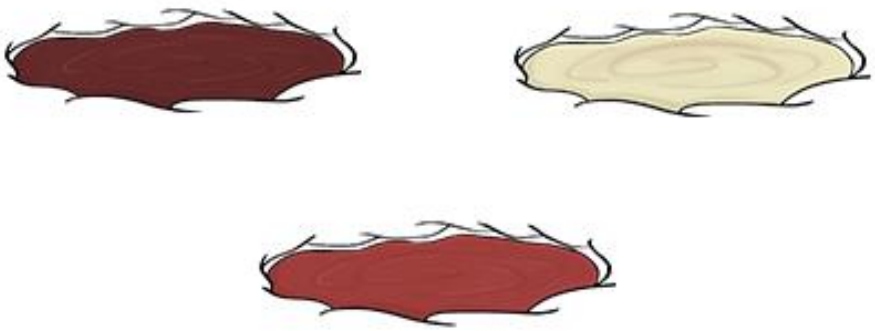

Figura 44: *Salsas para pizzas. De izquierda a derecha y de arriba a abajo: barbacoa, bechamel y tomate.*
Fuente: Elaboración propia.

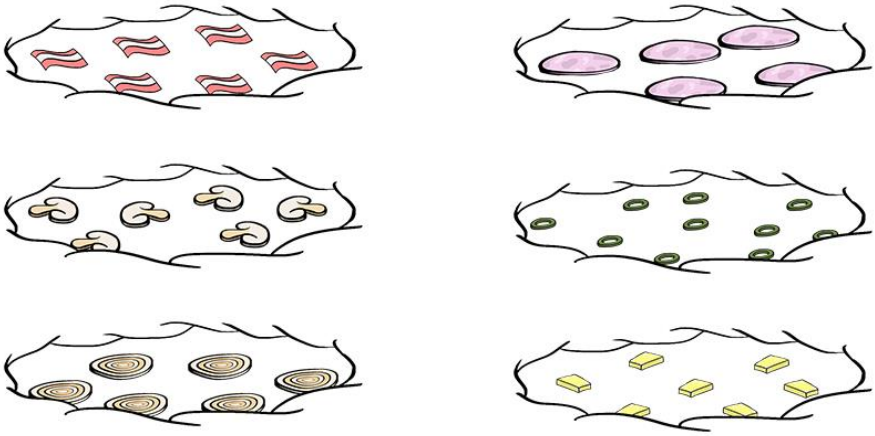

Figura 45: *Aderezos para pizzas. De izquierda a derecha y de arriba abajo: beicon, jamón, champiñones, olivas, cebolla y piña.*
Fuente: Elaboración propia

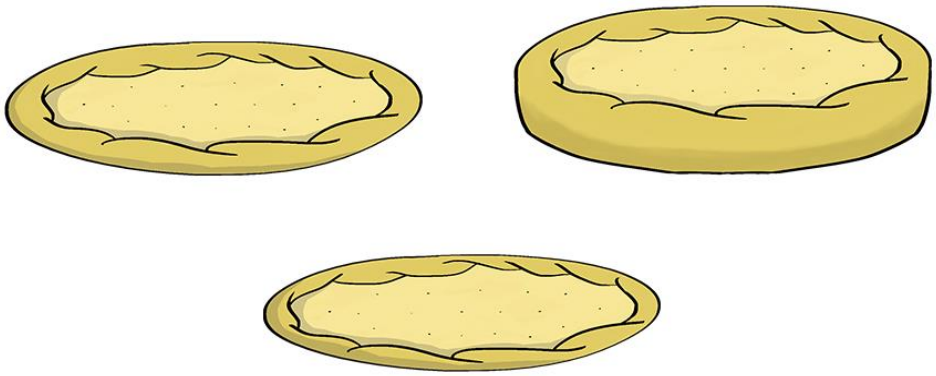

Figura 46: *Masas de pizzas. De izquierda a derecha y de arriba a abajo: normal, gruesa y fina.*
Fuente: Elaboración propia

.

5.2. Sistema web

En esta sección se define el proceso de desarrollo del sistema web, desde el concepto hasta el producto final.

5.2.1. Concepto e ideas

El concepto del sistema web ha sido desde el inicio de este proyecto una página con varios juegos a elegir, en el que si se pulsa en cualquiera de ellos se muestren las diferentes métricas que almacena dicho juego. Esa parte no se ha modificado, pero se han añadido mejoras para añadir, a esta página de métricas, un sistema de creación de variables, que en su concepto inicial se añadían desde un plugin de Unity, que, finalmente, se descartó.

Dicho sistema web, en su estado actual, necesita tener una base de datos, en la que almacenar toda la información que se recibe, tanto de los juegos como del mismo sistema. Para manejar dicha base de datos se ha desarrollado un servidor, el cual es manejado por un sistema API, es decir, mediante peticiones HTTP al mismo. Esta API es llamada desde, tanto el sistema web, como desde los videojuegos, tanto para consultar información, como para crear nueva.

5.2.2. Diseño de la base de datos

Del diseño de la base de datos depende el resto del sistema normalmente, y este proyecto no iba a ser menos. Como se especificó en la sección de herramientas del capítulo 4, la herramienta usada para este sistema, y para la que hubo que amoldar el diseño, es MySQL.

El primer paso para el diseño de una base de datos en MySQL es definir qué clases van a intervenir en el sistema. En este caso, a priori, se tendría la clase juego, la clase métrica y la clase variable. La clase juego tendría múltiples métricas, pero una métrica solo podría estar relacionada con un juego. Además, métricas tendría que tener múltiples variables, y las variables podrían relacionarse con muchas métricas. Por tanto, el esquema quedaría como en la Figura 47.

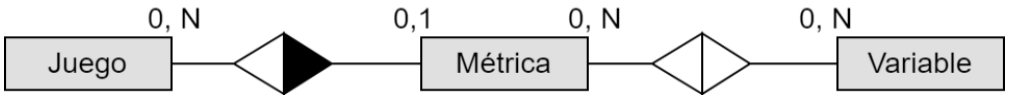

Figura 47: *Primer diagrama de la base de datos.*
Fuente: Elaboración propia.

Este diagrama tiene un problema, ya que debido a la fuente de los datos, múltiples usuarios desde los videojuegos, se tendrían miles de variables disponibles para asignar a las métricas, sin tener en cuenta los usuarios, o partidas, por lo que se decidió añadir dichas clases al modelo, además de varias clases extras, para almacenar las variables que vayan aportando los jugadores, para almacenar las partidas y para almacenar los jugadores, por lo que el diagrama de dicha base de datos queda tal y como se muestra en la Figura 48.

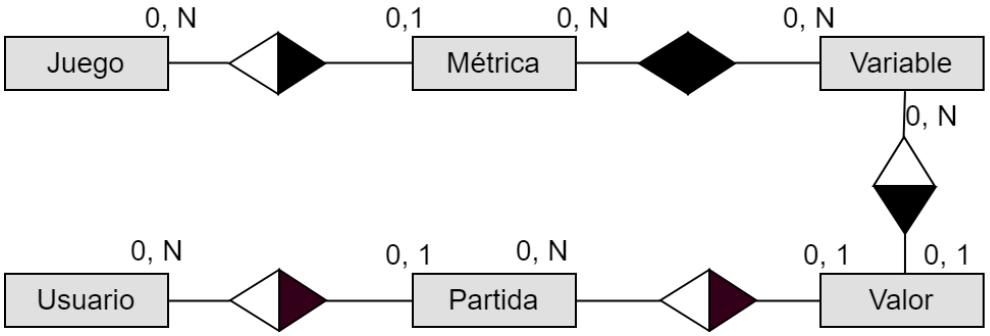

Figura 48: *Diagrama final de la base de datos.*
Fuente: Elaboración propia.

Una vez con este diagrama, el siguiente paso es decidir qué atributos tendrá cada clase y posteriormente pasarlo a SQL, el lenguaje usado por MySQL.

5.2.3. Desarrollo del servidor

Respecto al desarrollo del servidor, se ha seguido la filosofía de diseño de rutas - controlador. Las rutas son el lugar donde se tendrá que hacer la petición por clase, es decir, la URL de la misma. Cada una de estas rutas llamará a una función del controlador de cada clase para realizar diferentes funcionalidades, como por ejemplo consultar un juego. Todas las clases tienen, en su controlador y ruta, funciones de consulta de un elemento en particular, de todos los elementos, además de creación, editado y borrado. Además de estas funciones, también hay otras que

ejecutan consultas personalizadas como, por ejemplo, dar todas las métricas de un juego, o todos los valores de una variable.

5.2.4. Desarrollo de la página web

Respecto a la página web, el diseño se ha basado en la sencillez y pensando siempre primero en el diseño móvil y luego adaptándolo al diseño en ordenador, y no al revés.

Dicho diseño se basa en una imagen grande por cada ítem que tengamos en la página, considerando como ítem tanto una métrica como un juego, dependiendo de la página web a la que nos refiramos. Esta interfaz se puede observar en la Figura 49 en una pantalla panorámica, y en móvil en la Figura 50.

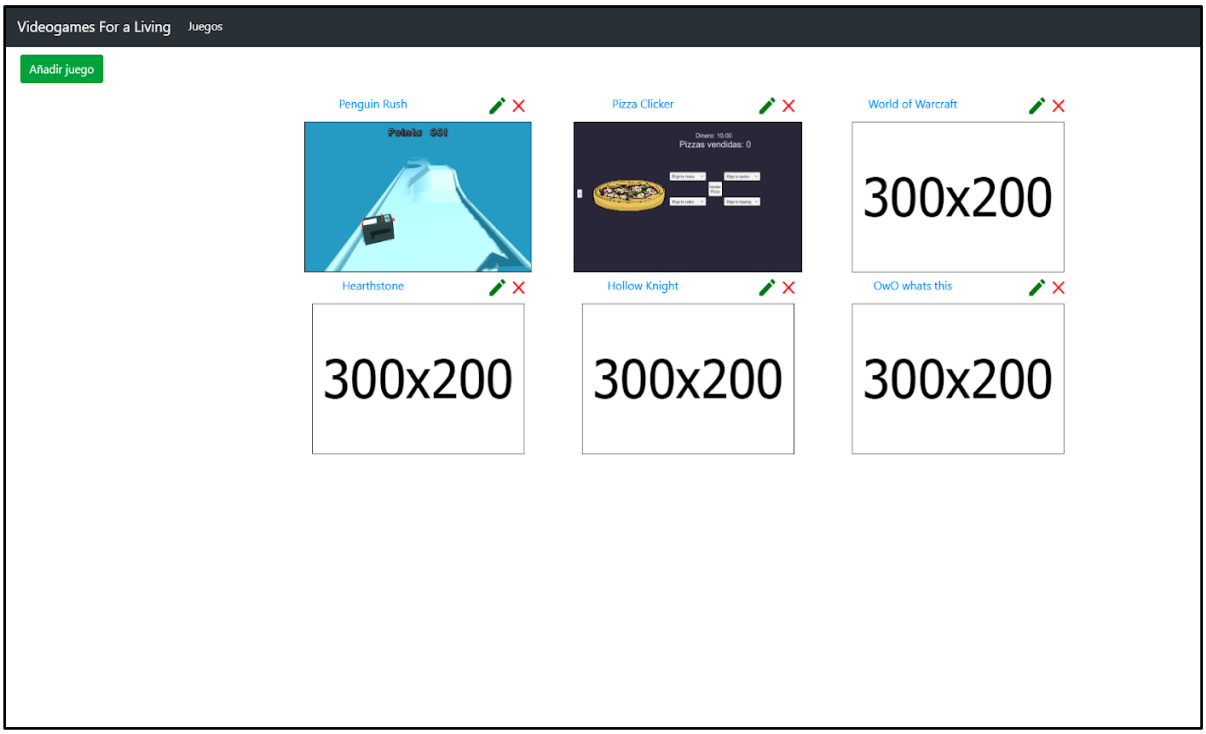

Figura 49: *Página juegos en pantalla panorámica.*
Fuente: Elaboración propia.

Figura 50: *Página juegos en pantalla de móvil.*
Fuente: Elaboración propia.

Respecto a la página de métricas, es muy similar a la página de juegos, como se puede observar en la Figura 51 y en la Figura 52.

Respecto a la lógica interna de la web se sigue el patrón por defecto y recomendado por Angular. Este patrón se basa en cuatro entidades interrelacionadas, el modelo, el controlador, el servicio y la vista, aunque normalmente esta se engloba en el controlador.

El modelo es el lugar donde se definen los diferentes atributos que puede tener un objeto, como en nuestro proyecto, un juego. El servicio se encarga de conectar con el servidor mediante conexiones HTTP y es responsable de recibir la información cruda del mismo. El controlador tiene como función preparar toda la información, tanto para enviar al servicio, como para formatearla adecuadamente para mostrarla al usuario. Por último, la vista es lo que verá el usuario final, es decir, la web en sí, por lo que nos encontraremos principalmente HTML y CSS en esta capa.

Figura 51: *Página métricas en pantalla panorámica.*
Fuente: Elaboración propia.

Para este proyecto, se han definido dos vistas, la vista de juegos y la vista de métricas.

Figura 52: *Página métricas en pantalla de móvil. Fuente: Elaboración propia.*

En la vista de juegos encontramos, por primera vez al entrar en la web, una barra superior, que aparecerá en todas las vistas de la web. En ella se incluyen todas las posibles secciones, aunque en este caso solo estará “Juegos”, ya que para acceder a métricas se ha de seleccionar antes un juego.

Además de esto, en esta página se encuentra un botón de “añadir juego” y los juegos ya creados. Pulsar este botón abrirá el modal de creación de juego, que se ve en la Figura 53.

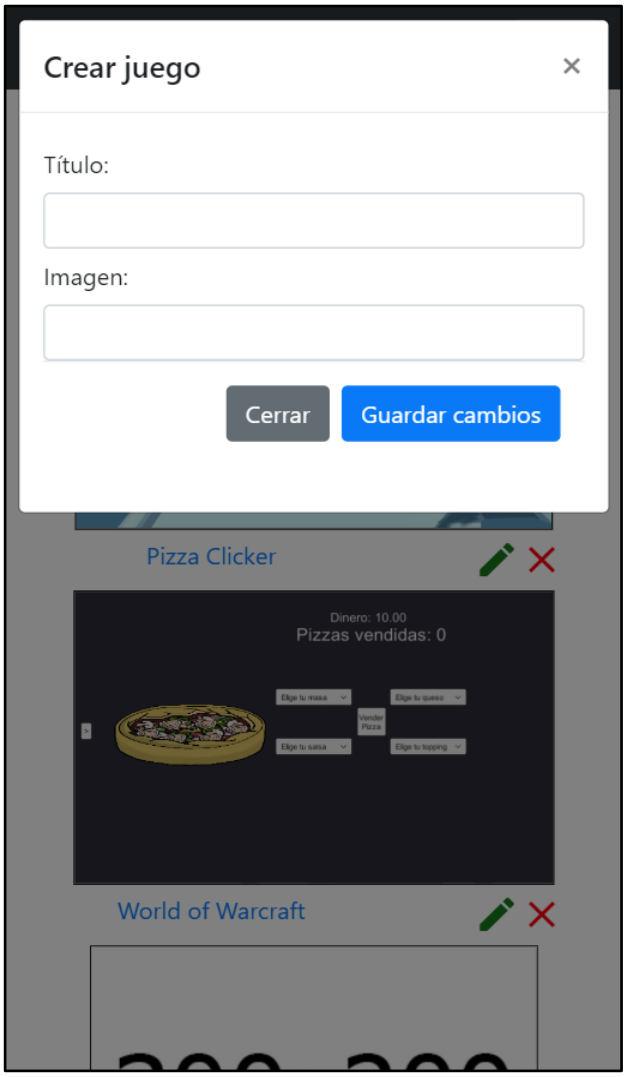

Figura 53: *Modal de creación de juegos en pantalla de móvil.*
Fuente: Elaboración propia.

Respecto a los juegos, el diseño es muy visual, ya que se prioriza la imagen respecto al texto. Pulsar cualquiera de estos dos elementos abrirá la página de dicho juego, y por tanto, la página métricas. Adicionalmente, también encontramos al lado del texto de cada videojuego dos botones, con un lápiz y una cruz. El botón del lápiz permitirá editar el juego, ya sea su nombre, o su imagen. Por último, al pulsar el botón de la cruz el juego se eliminaría de la base de datos, dando antes un aviso de seguridad. Esta estructura de creación / edición / borrado también se ha aplicado a la página de métricas de cada juego.

Respecto a la vista de métricas, se muestra una vista similar a la de juegos, pero en lugar de juegos, tendríamos métricas. Además, se añade un botón nuevo en la barra superior, añadir variables, que abre un modal que nos permite añadir tantas variables como queramos. Estas variables serán usadas posteriormente para crear métricas y para alimentarlas desde los videojuegos, por lo que son una parte verdaderamente importante en este proyecto. Dicho modal se muestra en la Figura 54. En esta figura se puede observar que en dicho modal se tiene una Id, el nombre de la variable, y 3 componentes, x, y, z, para poder realizar métricas en 2 e incluso 3 dimensiones. Estas variables se han usado para la ilustración de las métricas en la Figura 52, de manera que los tres primeros Ids pertenecen a la primera métrica y el resto a la segunda.

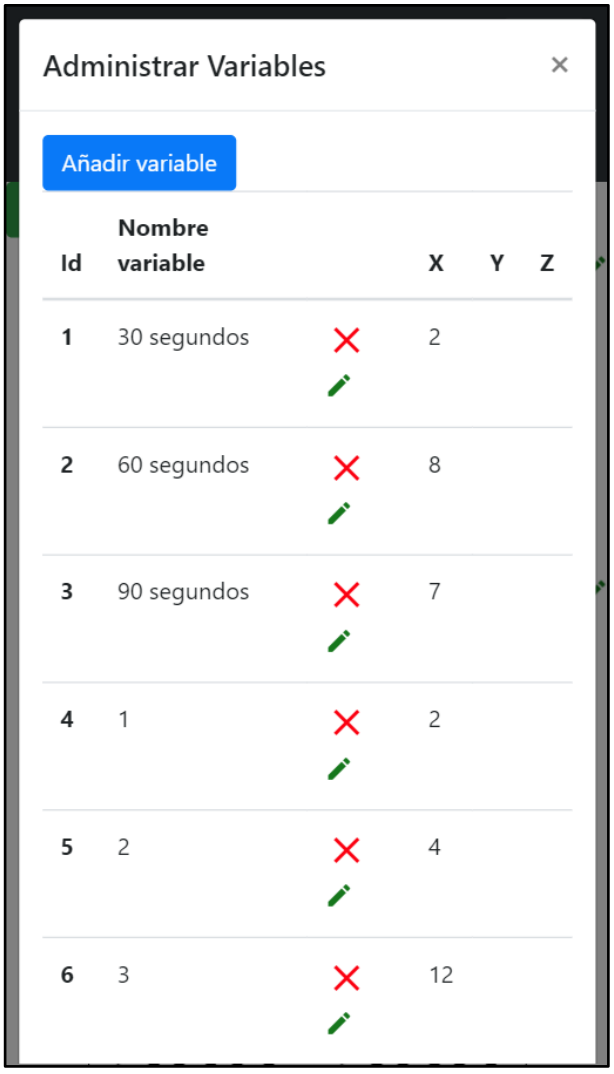

Figura 54: *Modal de creación de variables en pantalla de móvil.*
Fuente: Elaboración propia.

Respecto al modal de métricas, tiene muchos más campos que el de crear juegos, como por ejemplo el elegir qué tipo de métrica se quiere plasmar, qué variables va a tener dicha métrica, o de qué color va a ser cada barra, línea, o lo que queramos en este caso. Dicho modal se muestra en la Figura 55.

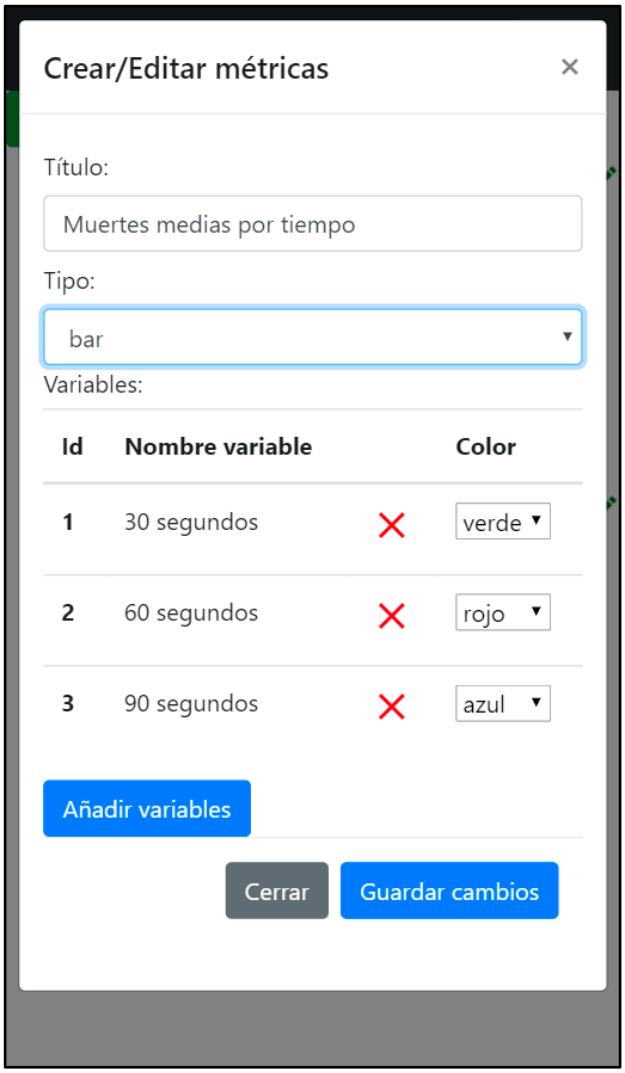

Figura 55: *Modal de creación de métricas en pantalla de móvil.*
Fuente: Elaboración propia.

6. Conclusiones y líneas futuras

En conclusión, este proyecto ha cumplido en prácticamente su totalidad sus objetivos: los videojuegos realizados, *Pizza Clicker* y *Penguin Rush*, y el sistema web de métricas para dichos juegos.

Además, durante este desarrollo, pese a tener conocimientos previos, tanto en diseño y desarrollo de videojuegos, servidores y webs, he aprendido a usar mucho mejor las herramientas citadas en el Capítulo 4.

Respecto a la planificación, la mayoría ha ido de acuerdo a la propuesta inicial, pero han surgido problemas con ciertos aspectos, entre los que se destacan las peticiones HTTP y la comunicación entre sistemas. Además, he necesitado más tiempo del estimado para la elaboración de la memoria, especialmente para la revisión y maquetación de la misma. Ya que, aunque en un principio se pensó realizarla completamente con documentos de Google Drive, ante la falta de ciertas funcionalidades, especialmente relacionadas con las referencias cruzadas y los índices, se decidió hacer la maquetación final con Microsoft Word.

Como línea futura, para este proyecto, sería interesante incluir un sistema de seguridad. Este sistema de seguridad se plantearía de manera que tuviera un sistema en el servidor por el que pasarían todas las peticiones que puedan afectar al sistema. Dichas peticiones serían controladas por este sistema, mediante un sistema de tokens, los cuales se generarían con un usuario y contraseña, y tendrían un tiempo de vida limitado.

Por otra parte también se podría añadir soporte multilenguaje tanto a Pizza Clicker como al sistema web. Para Pizza Clicker se podría utilizar la utilidad de localización de los textos en Unity. De esta forma se tendría una lista maestra con todos los textos, y cambiar el idioma sería tan fácil como cambiar ese archivo a otro en otro idioma. Para el sistema web se debería usar un sistema similar, es decir, archivos con todos los textos del sistema en diferentes idiomas.

Además de esto, se podría añadir al sistema web la posibilidad de comparar métricas, es decir, superponer una encima de otra, de manera que se puedan comparar las mismas, y entender qué información se nos da, y, añadir un sistema de seguimiento a los usuarios, para poder encajarlos en diferentes perfiles de jugador.

Por último decir que la realización de este proyecto me ha enseñado cómo realizar un proyecto grande y me ha permitido aplicar conceptos de Ingeniería de Software que, hasta el momento, solo había visto en el ámbito teórico.

BIBLIOGRAFÍA

[1] Ace Attorney. Capcom, 2017. Web oficial del videojuego: http://www.ace-attorney.com/.

[2] A Hat in Time. Gears for breakfast, 2017. Web oficial del videojuego: http://hatintime.com/.

[3] Amnesia. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 15 de octubre de 2018]. https://en.wikipedia.org/wiki/Amnesia:_The_Dark_Descent.

[4] Angular 6. Google, 2018. Disponible en: https://angular.io/.

[5] Arma: Armed Assault. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 15 de octubre de 2018]. Disponible en: https://es.wikipedia.org/wiki/ArmA:_Armed_Assault.

[6] Arrioja, N. Unity – Diseño y programación de videojuegos, Redusers, Buenos Aires 2013.

[7] ASP.NET. Microsoft, 2002-2015. Disponible en: https://www.asp.net/.

[8] Autodesk 3DS Max. Autodesk Inc., 2018. Disponible en: https://www.autodesk.es/products/3ds-max/overview.

[9] Autodesk Maya. Autodesk Inc., 2017. Disponible en: https://www.autodesk.es/products/maya/overview.

[10] Benzo, F., González, A., Litle S. y otros. Anuario de la industria del videojuego. Asociación Española de Videojuegos, 2017. Disponible en: http://www.aevi.org.es/web/wp-content/uploads/2018/07/AEVI_Anuario2017.pdf

[11] Bitbucket. Atlassian, 2008. Disponible en: https://bitbucket.org/.

[12] Blender 3D. Blender Foundation, 2018. Disponible en: https://www.blender.org/.

[13] Bootstrap 4. Twitter, 2011-2018. Disponible en: https://getbootstrap.com/.

[14] Candy Crush Saga. King, 2012. Web oficial del videojuego: https://king.com/es/game/candycrush.

[15] Castlevania. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 15 de octubre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Castlevania.

[16] Civilization (serie). Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/Civilization_(serie).

[17] Contreras, I. ¿Qué es la disonancia ludonarrativa? Freakelitex, 4 de febrero de 2018. Disponible en: http://freakelitex.com/la-disonancia-ludonarrativa/.

[18] Cookie Clicker. Julien Thiennot, 2013. Web oficial del videojuego: http://orteil.dashnet.org/cookieclicker/.

[19] Cuphead. StudioMDHR, 2017. Web oficial del videojuego: http://www.cupheadgame.com/.

[20] Detroit Become Human. Quantic Dream, 2018. Web oficial del videojuego: https://www.playstation.com/en-us/games/detroit-become-human-ps4/.

[21] Diablo (videojuego). Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 15 de octubre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Diablo_(videojuego).

[22] Dungeons & Dragons (Dragones y Mazmorras). Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 15 de octubre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Dungeons_%26_Dragons.

[23] Esposito, N. A short and simple definition of what a videogame is. In Proceedings of DiGRA (Digital Games Research Association) conference: Changing views – Worlds in play, 2005.

[24] Euro Truck Simulator. SCS Software. 2008-2016. Web oficial del videojuego: https://eurotrucksimulator.com/.

[25] Fallout 4. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 15 de octubre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Fallout_4.

[26] Final Fantasy VII, Square Enix, 1997. Web oficial del videojuego: https://finalfantasyviipc.square-enix-games.com/.

[27] Fortnite. Epic Games, 2017. Web oficial del videojuego: https://www.epicgames.com/fortnite/es-ES/home.

[28] GameMaker: Studio. YoYo Games, 2017. Disponible en: http://www.yoyogames.com/.

[29] Gears of War. Epic Games, 2016. Web oficial del videojuego: https://gearsofwar.com/es-es.

[30] GitHub. GitHub, Inc., 2010. Disponible en: https://github.com/.

[31] GitLab. GitLab Inc., 2011. Disponible en: https://about.gitlab.com/.

[32] Grand Theft Auto. David Jones, Sam Houser, Dan Houser, 1997-2013. Web oficial del videojuego: http://www.rockstargames.com/grandtheftauto/.

[33] Guacamelee. DrinkBox Studios, 2013. Web oficial del videojuego: http://guacamelee.com/.

[34] Half Life. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 15 de octubre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Half-Life.

[35] Haverbeke, M. Eloquent JavaScript. 3rd edition, 2018. Disponible en: http://eloquentjavascript.net/.

[36] Hearthstone. Blizzard Entertainment, 2014. Web oficial del videojuego: https://playhearthstone.com/es-es/.

[37] Holgado, E. Métricas que todo desarrollador de videojuegos debe conocer. laSalle, Universidad Ramón Llull, 29 de julio de 2015. Disponible en: http://grados.salleurl.edu/multimedia/blog/metricas-desarrollador-videojuegos-debe-conocer/.

[38] Hollow Knight. Team Cherry, 2017. Web oficial del videojuego: http://hollowknight.com.

[39] Hyper Light Drifter. Heart Machine, 2016. Web oficial del videojuego: http://www.heart-machine.com/.

[40] JavaScript. Pluralsight, 2016-2018. Disponible en: https://www.javascript.com/.

[41] Jquery. The jQuery Foundation, 2006-2017. Disponible en: https://jquery.com/.

[42] Kingdom Hearts. Square Enix, 2002-2019. Web oficial del videojuego: http://www.kingdomhearts.com/home/es/.

[43] Larman, C. UML y patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado, Pearson Educación, S.A., Madrid 2003.

[44] League of Legends. Riot Games Inc., 2009. Web oficial del videojuego: https://play.euw.leagueoflegends.com/es_ES.

[45] List of most expensive video games to develop. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 6 de octubre de 2018]. Disponible en: https://en.wikipedia.org/wiki/List_of_most_expensive_video_games_to_develop.

[46] List of video game genres. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. Disponible en: https://en.wikipedia.org/wiki/List_of_video_game_genres.

[47] Los Sims. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Los_Sims.

[48] Marqués, R. Estudio comparativo de software de diseño de gráficos en tres dimensiones. Trabajo Fin de Grado. Universidad de Valladolid, 2015.

[49] Martí-Parreño, J. Marketing y videojuegos: Product placement, in-game advertising y advergaming, ESIC Editorial, Madrid 2010.

[50] Metal Slug. SNK Playmore, 1996. Web oficial del videojuego: http://game.snk-corp.co.jp/official/metalslug_sp/index.html.

[51] Método MoSCoW. tic.PORTAL, 11 de junio de 2018. Disponible en: https://www.ticportal.es/glosario-tic/metodo-moscow.

[52] Metroid. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Metroid.

[53] Minecraft. Mojang AB, 2009-2018. Web oficial del videojuego: .https://minecraft.net/es-es/.

[54] MongoDB. MongoDB, Inc., 2009. Disponible en: https://www.mongodb.com/es.

[55] Monkey Island. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Monkey_Island.

[56] Mula, J. Pros y contras de programar en Unity vs. en Unreal Engine. Deusto formación, 21 de noviembre de 2017. Disponible en: https://www.deustoformacion.com/blog/diseno-produccion-audiovisual/pros-contras-programar-unity-vs-unreal-engine.

[57] MySQL. Oracle Corporation, 1995-2018. Disponible en: https://www.mysql.com/.

[58] Node.js. Node.js Foundation, 2009-2018. Disponible en: https://nodejs.org/es/.

[59] Orcs must die! Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://en.wikipedia.org/wiki/Orcs_Must_Die!

[60] Ori and the Blind Forest. Moon Studios, 2015. Web oficial del videojuego: https://www.orithegame.com/.

[61] Overwatch. Blizzard Entertainment Inc., 2018. Web oficial del videojuego: https://playoverwatch.com/es-es/.

[62] Pereira, F., Alonzo, T. Hacia una conceptualización de los videojuegos como discursos multimodales electrónicos. Anagramas: Rumbos y sentidos de la comunicación, 15(30):51-64, 2017.

[63] Photoshop. Adobe Systems Incorporated, 2018. Disponible en: https://www.adobe.com/es/products/photoshop.html.

[64] PHP. Rasmus Lerdorf, 1995-2018. Disponible en: http://php.net/.

[65] Portal (videojuego). Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/Portal_(videojuego).

[66] Pros y contras de usar un motor de juegos, WiMi5 Inc., 12 de junio de 2015. Disponible en: http://wimi5.com/pros-y-contras-de-usar-un-motor-de-juegos/.

[67] React. Facebook Inc., 2018. Disponible en: https://reactjs.org/.

[68] Resident Evil. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/Resident_Evil.

[69] Rogue. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/Rogue.

[70] Ruby. Yukihiro Matsumoto, 1995-2017. Disponible en: https://www.ruby-lang.org/es/.

[71] Scholand, M. Localización de videojuegos. Tradumàtica, 1, 2002. Disponible en: ttps://ddd.uab.cat/pub/tradumatica/15787559n1/15787559n1a4scholand.pdf.

[72] Schreier, J. Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made, Jason Harper, New York 2017.

[73] Servidor HTTP Apache. Apache Software Foundation, 1995-2017. Disponible en: https://www.apache.org/.

[74] Shahrani, S. Educational feature: A history and analysis of level design in 3D computer games. Gamasutra, 25 de abril de 2016. Disponible en: http://www.gamasutra.com/view/feature/2674/educational_feature_a_history_and_.php.

[75] SimCity. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/SimCity.

[76] Snow Bros, Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/Snow_Bros.

[77] Starcraft. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/StarCraft.

[78] Street Fighter II: The World Warrior. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/Street_Fighter_II:_The_World_Warrior.

[79] Suau, P. Manual de modelado y animación con Blender. Servicio de Publicaciones de la Universidad de Alicante, Alicante 2011.

[80] Super Mario Bros. Nintendo Entertainment System, 1985. Web oficial del videojuego: www.nintendo.es/Juegos/NES/Super-Mario-Bros--803853.html.

[81] Super Smash Bros. Nintendo, 1999-2018. Web oficial del videojuego: https://www.smashbros.com/en_US/.

[82] Tetris. Tetris Holding, 1985-2018. Web oficial del videojuego: https://tetris.com/.

[83] The Stanley Parable. Galactic Cafe, 2013. Web oficial del videojuego: https://stanleyparable.com/

[84] The Wolf Among Us. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/The_Wolf_Among_Us.

[85] Undertale. Toby Fox, 2015-2018. Web oficial del videojuego: https://undertale.com/.

[86] Unity. Unity Technologies, 2018. Disponible en https://unity3d.com/es.

[87] Unity vs Unreal – Una comparación teórica. University of Advanced Technologies, 25 de septiembre de 2017. Disponible en: https://www.uniat.com/unity-vs-unreal-comparacion-teorica/.

[88] Unreal Engine. Epic Games Inc., 2004-2018. Disponible en: https://www.unrealengine.com.

[89] Unreal. Epic Games, 1998-2007. Web oficial del videojuego: https://www.epicgames.com/unrealtournament/.

[90] Vasquez, W. Unity vs Unreal ¿Cuál es el Mejor Motor Gráfico para Crear Videojuegos? La cueva del lobo, 14 de junio de 2017. Disponible en: https://www.cuevadelobo.com/unity-vs-unreal-mejor-crear-videojuegos/.

[91] Videojuego. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 6 de julio de 2018]. Disponible en: https://es.wikipedia.org/wiki/Videojuego.

[92] World Of Warcraft. Blizzard Entertainment, 2004-2018. Web oficial del videojuego: https://worldofwarcraft.com/es-es/.

[93] Zork. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 20 de septiembre de 2018]. https://es.wikipedia.org/wiki/Zork.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS